JC03 Rec'd PCT/PTO 23 OCT 2001

FORM PTO-1390 (Modified) (REV 11-2000) U.S. DEPARTMENT OF COMMERCE PATENT AND TRADEMARK OFFICE

TRANSMITTAL LETTER TO THE UNITED STATES DESIGNATED/ELECTED OFFICE (DO/EO/US) CONCERNING A FILING UNDER 35 U.S.C. 371

ATTORNEY'S DOCKET NUMBER: **1419-136 US**

U.S. APPLICATION NO. (IF KNOWN, SEE 37 CFR): **10/031260**

INTERNATIONAL APPLICATION NO.	INTERNATIONAL FILING DATE	PRIORITY DATE CLAIMED
PCT/US00/11428	**28 April 2000 (28.04.00)**	**29 April 1999 (29.04.99)**

TITLE OF INVENTION
Distributed Software Development Environment

APPLICANT(S) FOR DO/EO/US
Manjunath SURYANARAYANA

Applicant herewith submits to the United States Designated/Elected Office (DO/EO/US) the following items and other information:

1. ☒ This is a **FIRST** submission of items concerning a filing under 35 U.S.C. 371.
2. ☐ This is a **SECOND** or **SUBSEQUENT** submission of items concerning a filing under 35 U.S.C. 371.
3. ☐ This is an express request to begin national examination procedures (35 U.S.C. 371(f)). The submission must include itens (5), (6), (9) and (24) indicated below.
4. ☐ The US has been elected by the expiration of 19 months from the priority date (Article 31).
5. ☒ A copy of the International Application as filed (35 U.S.C. 371 (c) (2))
   a. ☒ is attached hereto (required only if not communicated by the International Bureau).
   b. ☐ has been communicated by the International Bureau.
   c. ☐ is not required, as the application was filed in the United States Receiving Office (RO/US).
6. ☐ An English language translation of the International Application as filed (35 U.S.C. 371(c)(2)).
   a. ☐ is attached hereto.
   b. ☐ has been previously submitted under 35 U.S.C. 154(d)(4).
7. ☐ Amendments to the claims of the International Application under PCT Article 19 (35 U.S.C. 371 (c)(3))
   a. ☐ are attached hereto (required only if not communicated by the International Bureau).
   b. ☐ have been communicated by the International Bureau.
   c. ☐ have not been made; however, the time limit for making such amendments has NOT expired.
   d. ☐ have not been made and will not be made.
8. ☐ An English language translation of the amendments to the claims under PCT Article 19 (35 U.S.C. 371(c)(3)).
9. ☒ An oath or declaration of the inventor(s) (35 U.S.C. 371 (c)(4)).
10. ☐ An English language translation of the annexes to the International Preliminary Examination Report under PCT Article 36 (35 U.S.C. 371 (c)(5)).
11. ☐ A copy of the International Preliminary Examination Report (PCT/IPEA/409).
12. ☒ A copy of the International Search Report (PCT/ISA/210).

Items 13 to 20 below concern document(s) or information included:

13. ☐ An Information Disclosure Statement under 37 CFR 1.97 and 1.98.
14. ☐ An assignment document for recording. A separate cover sheet in compliance with 37 CFR 3.28 and 3.31 is included.
15. ☐ A **FIRST** preliminary amendment.
16. ☐ A **SECOND** or **SUBSEQUENT** preliminary amendment.
17. ☐ A substitute specification.
18. ☐ A change of power of attorney and/or address letter.
19. ☐ A computer-readable form of the sequence listing in accordance with PCT Rule 13ter.2 and 35 U.S.C. 1.821 - 1.825.
20. ☐ A second copy of the published international application under 35 U.S.C. 154(d)(4).
21. ☐ A second copy of the English language translation of the international application under 35 U.S.C. 154(d)(4).
22. ☒ Certificate of Mailing by Express Mail
23. ☐ Other items or information:

U.S. APPLICATION NO. (IF KNOWN, SEE 37 CFR)	INTERNATIONAL APPLICATION NO.	ATTORNEY'S DOCKET NUMBER
10/031260	PCT/US00/11428	1419-136 US

24. The following fees are submitted:.		CALCULATIONS PTO USE ONLY	
BASIC NATIONAL FEE (37 CFR 1.492 (a) (1) - (5)) :			
☐ Neither international preliminary examination fee (37 CFR 1.482) nor international search fee (37 CFR 1.445(a)(2)) paid to USPTO and International Search Report not prepared by the EPO or JPO	**$1040.00**		
☐ International preliminary examination fee (37 CFR 1.482) not paid to USPTO but International Search Report prepared by the EPO or JPO	**$890.00**		
☐ International preliminary examination fee (37 CFR 1.482) not paid to USPTO but international search fee (37 CFR 1.445(a)(2)) paid to USPTO	**$740.00**		
☒ International preliminary examination fee (37 CFR 1.482) paid to USPTO but all claims did not satisfy provisions of PCT Article 33(1)-(4)	**$710.00**		
☐ International preliminary examination fee (37 CFR 1.482) paid to USPTO and all claims satisfied provisions of PCT Article 33(1)-(4)	**$100.00**		
ENTER APPROPRIATE BASIC FEE AMOUNT =		**$710.00**	
Surcharge of **$130.00** for furnishing the oath or declaration later than ☐ 20 ☒ 30 months from the earliest claimed priority date (37 CFR 1.492 (e)).		**$130.00**	

CLAIMS	NUMBER FILED	NUMBER EXTRA	RATE		
Total claims	58 - 20 =	38	x $18.00	**$684.00**	
Independent claims	9 - 3 =	6	x $84.00	**$504.00**	
Multiple Dependent Claims **(check if applicable)**.			☐	**$0.00**	
TOTAL OF ABOVE CALCULATIONS			=	**$2,028.00**	
☒ Applicant claims small entity status. See 37 CFR 1.27). The fees indicated above are reduced by 1/2.				**$1,014.00**	
SUBTOTAL			=	**$1,014.00**	
Processing fee of **$130.00** for furnishing the English translation later than ☐ 20 ☐ 30 months from the earliest claimed priority date (37 CFR 1.492 (f)).			+	**$0.00**	
TOTAL NATIONAL FEE			=	**$1,014.00**	
Fee for recording the enclosed assignment (37 CFR 1.21(h)). The assignment must be accompanied by an appropriate cover sheet (37 CFR 3.28, 3.31) **(check if applicable).** ☐				**$0.00**	
TOTAL FEES ENCLOSED			=	**$1,014.00**	
				Amount to be: refunded	$
				charged	$

a. ☒ A check in the amount of **$1,014.00** to cover the above fees is enclosed.

b. ☐ Please charge my Deposit Account No. _____ in the amount of _____ to cover the above fees. A duplicate copy of this sheet is enclosed.

c. ☒ The Commissioner is hereby authorized to charge any additional fees which may be required, or credit any overpayment to Deposit Account No. **13-2165** A duplicate copy of this sheet is enclosed.

d. ☐ Fees are to be charged to a credit card. **WARNING:** Information on this form may become public. **Credit card information should not be included on this form.** Provide credit card information and authorization on PTO-2038.

NOTE: Where an appropriate time limit under 37 CFR 1.494 or 1.495 has not been met, a petition to revive (37 CFR 1.137(a) or (b)) must be filed and granted to restore the application to pending status.

SEND ALL CORRESPONDENCE TO:

Diane Dunn McKay, Esq.
Mathews, Collins, Shepherd & Gould, P.A.
100 Thanet Circle, Suite 306
Princeton, NJ 08540
(609) 924-8555 Telephone
(609) 924-3036 Facsimile

SIGNATURE

Diane Dunn McKay
NAME

34,586
REGISTRATION NUMBER

October 22, 2001
DATE

23 OCT 2001

DISTRIBUTED SOFTWARE DEVELOPMENT ENVIRONMENT

This application claims priority of Provisional Application No. 60/131,506 entitled "Distribution Software Development Environment" the entire contents of which are hereby incorporated by reference into this application.

Background of the Invention

1. Field of the Invention

The present invention relates to a distributed software environment for software development and maintenance.

2. Description of the Related Art

The engineering of large scale evolving software systems is difficult in traditional linear programming code because of the complexity that is needed to design and maintain such systems. Also, minor changes to the software in response to user demands can require major redesign and rewriting of the entire program. Object-oriented programming techniques have been used to reduce software development complexity. A description of problems associated with engineering of large-scale evolving software and an overview of possible solutions to the problem is described by the inventor in "OMSOFT: A Change Management Paradigm," Journal of Network and Systems Management, Vol. 6, No. 1, 1998.

In object-oriented programming, an object is typically defined as a combination of data and operations to be performed on that data. An application network is generally a collection of the objects(programs) interconnected by a common interface in order to perform a given task. Object-oriented software systems typically provide an architecture specification, referred to as an object model, which enables developed objects to work together in an application.

An industry consortium called the Object Management Group ("OMG") has developed a standard for object-oriented systems under the Common Object Request Broker Architecture ("CORBA") specification. The CORBA specification defines the distributed computing environment in terms of objects in a distributed client-server environment. An object broker of CORBA provides a mechanism by which objects or other applications can make requests to and

receive responses from other objects managed by the system. The CORBA approach makes use of multiple object brokers, with the various components of the system distributed, as needed, to accommodate growth or the reassignment of system resources. Each of these multiple object brokers is capable of accessing information in corresponding implementation and interface repositories. Object requests are transparent to the users in that an object operation can be performed on an array of several computing devices and the entity invoking the object operation need not be aware of where the object operation resides or is executed. Consequently, by using the CORBA approach, the individual user is freed from the responsibility of administrating where specific programs reside or where they are executed since these details are handled by the administrators of the CORBA-based system. In addition, by defining a fixed set of operation, the CORBA approach enables object brokers from different vendors to communicate with each other, causing the system to be more universally compatible.

New objects can be added to a CORBA-based system simply by updating the information in the respective repositories since the information on how to interface to an object and execute the object are contained in the interface and implementation repositories. Thus, the CORBA based system can be upgraded easily to accommodate new or modified objects. Because the object-oriented approach allows the actual implementation of an object's operations to be transparent to the user, existing applications need only be modified in their interface in order to function in a CORBA based system. Consequently, existing applications need not be completely rewritten in order to work with the system.

U.S. Patent No. 5,838,970 describes that CORBA has several limitations. First, CORBA does not provide a mechanism for moving objects after an object is created. Second, CORBA does not make an individual user's information accessible throughout the network or permit the system resources to be administered at either the user level or the object level. In addition, CORBA does not support the use of multiple versions of a given object type. CORBA also does not provide for an effective means of duplicating highly used resources. The '970 patent describes an object-oriented computer environment in which the manner of processing or transmitting information can be altered during the course of operation. The environment is managed by storing in a plurality of repositories accessible during life cycles of objects, and information required to initiate object operations. The repositories can be assigned different levels of priority to control the distribution of stored information. By using information retrieved

from changeable repositories to affect object operations, computer based information processing can be expanded as needed. Also, software upgrades can be made by changing the contents of the application repositories. CORBA has the additional shortcomings of not providing for selection and resolution of shared variables, selection of compatible application systems and dynamic reconfiguration of reusable CORBA components, and CORBA supports a single interface.

The CORBA facility is in its infancy since it has failed to address systems management issues discussed before (with respect to large network-oriented applications). In addition, it is noted that CORBA was designed without considering the impact of implementing systems (that manage networks and support the operations and services of the network provider) on the object and the interaction models. For example, CORBA blocks the sender of a message until the receiver receives and acknowledges it. This strains concurrency of distributed applications. Also there could be deadlocks between the sender and the recipient. Also, CORBA does not support multi-cycle transactions and does not support versioning of objects and interfaces. In addition, CORBA does not address buffer losses associated with asynchronous communication. Thus, programmers have to handle possible buffer losses as part of the structure and protocol associated with shared interfaces thus increasing object interface complexity. Also, this makes interface descriptions of asynchronous CORBA components using CORBA IDL very complicated.

Conventional methods of installing newer versions of objects or interfaces in a software application is to shut down the application and install the newer versions. This method can introduce an unacceptable delay during which the application is denied to the users of the software. A human manager can check the versions to determine if they are compatible. Incompatibility of objects can result from revisions in at least one of the interfaces of the object to make the interfaces incompatible. A human manager can manually find out the compatible systems and choose which one to run. Manual techniques have been described by Rakesh Agraway, and H.V. Jagadish in "On Correctly Configuring Versioned Objects," Proc. of the 15th. Intl. Conf. on VLDBs, 1989. However, if an object has a large number of versions it can be difficult for a human manager to determine compatible objects.

Conventional automated techniques use a rule-based approach to maintain compatible application systems, for example see *R.H. Katz, and E. Chang,* "Managing Change in a Comp-

Aided Design Database," Proc. of the 13th VLDB Conf., 1987. However in large network applications the structure and protocols on different interfaces is complex and difficult to represent in rules. Accordingly, it is desirable to provide an object oriented software development environment supporting dynamic updating of newer versions of the application, dynamic maintenance of compatible systems and dynamic reconfiguration while the application is executing thus eliminating shutdowns and facilitating interface evolution.

Summary of the Invention

The present invention relates to a distributed object-oriented software development environment. Objects for performing object operations can communicate to one another if associated interfaces are compatible. The environment provides sequential flow of control and data to the objects.

Each interface of an object has a unique global identification value. Compatible interfaces are determined by registering the global identification and version of the interfaces with a management framework. Preferably, the management framework assigns object versions and interface versions in an increasing manner. A distributed selection method can be used by management framework to generate a partial order of all unique compatible interface application systems.

The modified CORBA IDL framework fundamentally transforms the design process by facilitation semantically precise, and meaningful object interfaces that are independent of object coordination problems. The environment simplifies description of complex dependent/independent transactions in network oriented applications. Capabilities supported for the human manager, like abilities to read/write states of objects, get/set states of objects, ability to get/set states of clocks from outside without the participation of components, ability to convert the raw data into meaningful data by getting type details from the modified CORBA IDL description and then placing the meaningful description inside the transaction using the modified CORBA IDL framework, facilitate the external observation and control of applications.

The environment can be used for establishing an application network and performing consistent and transparent dynamic updates of the application network. The management framework can include management objects associated with objects of the application. The management objects communicate information about the objects to a human manager object.

Accordingly, the human management object can control transactions of the objects during establishment or reconfiguration of the application network without participation by the objects. During a design stage, the management framework can also be used by the software developers for performing tetherless negotiation.

The invention will be more fully described by reference to the following drawings.

Brief Description of the Drawings

Fig. 1 is an architecture diagram of a distributed software development environment in accordance with the teachings of the present invention.

Fig. 2A is a schematic diagram of a distributed network of multiple objects.

Fig. 2B is a schematic diagram of a distributed network of multiple objects in a two phase transaction.

Fig. 3A is a schematic diagram of an interaction network and phases.

Fig. 3B is a schematic diagram of steps in different phases in an interaction clock.

Fig. 4 is a schematic diagram of a distributed management network.

Fig. 5 is a schematic diagram of stages of a life cycle.

Fig. 6 is a schematic diagram of a network application.

Fig. 7 is a flow diagram of a method of tetherless negotiation.

Fig. 8 is a schematic diagram of a negotiation network.

Fig. 9 is a block diagram of steps for implementing a network application.

Fig. 10 is a flow diagram of a method for determining compatible applications.

Fig. 11A is a schematic diagram of an evolving network application.

Fig. 11B is a schematic diagram of multiple compatible applications due to the evolution shown in Fig. 11A.

Fig. 12A is a schematic diagram illustrating a consistent and transparent network application set up by a human manager.

Fig. 12B is a schematic diagram illustrating minimization of disruption to achieve quiescence.

Fig. 13 is a flow diagram of a method for setting up a consistent and transparent application network.

Fig. 14 is a schematic diagram of the network application as shown in Fig. 11 during reconfiguration for replacement of an object.

Fig. 15. is a schematic diagram of the network application shown in Fig. 11 during reconfiguration for implementing a new version of an interface.

Detailed Description

Reference will now be made in greater detail to a preferred embodiment of the invention, an example of which is illustrated in the accompanying drawings. Wherever possible, the same reference numerals will be used throughout the drawings and the description to refer to the same or like parts.

Fig. 1 is a simplified architecture diagram of a distributed software development environment 10 in accordance with the teachings of the present invention. Environment 10 includes a plurality of frameworks. Each framework can either act independently or interact with one or more framework to design, implement, manage and maintain aspects of network oriented application 12. Distributed network framework 14 dynamically establishes objects and connections between the objects. Transaction framework 15 provides control of transactions between objects. Interaction framework 16 provides control of interactions between the objects. Modified COBRA IDL framework 18 provides object-oriented operations and an interface description language (IDL) for defining object interfaces. Human management framework 19 provides centralized management of information of objects and object interfaces. Life cycle framework 20 defines different stages of software development. Human dimension framework 21 defines human interaction with life cycle framework 20.

Distributed network 14 framework comprises an interactive network of objects that are connected by compatible interfaces. An object is an asynchronous processing entity that can initiate transaction requests to other objects and receive and service transaction requests from other connected objects. An object can be implemented as a collection of data structures and a set of methods to operate on the data structures. An object shares its interface with other objects through object ports. In a compatible interface the data structures are the same and the protocols are well understood by all object ports connected to that interface. Object ports can be dynamically created by the objects. Every object has a state. Its state is the collection of the values of a set of variables designated as state variables which totally characterize the object and

its runtime behavior. Transactions initiated by objects effect the states of corresponding receiving objects. Each object can be associated with a plurality of object ports. The state of a distributed network is comprised of internal states of constituent objects and states of interfaces between the objects.

Fig. 2A illustrates components of distributed network framework 14 in which object 30a includes object port 32a. Interface 34a connects to object port 32a. Interface 34a also connects to object port 32c of object 30c. Object port 32d of object 30c connects to interface 34b. Interface 34b also connects to object port 32b of object 30b. Accordingly, object 30a is connected via object 30c to object 30b. For example, object 30a can be a client or initiator object which initiates a transaction. Object 30c can be a receiver or server which receives proceedings and content of the transaction from object 30a. Object 30c can modify the received content.

Transaction framework 15 provides dynamic external observation and control using transaction phases. A transaction is a sequence of message exchanges between two or more objects, initiated by one of the objects. Accordingly, in the present invention, a transaction comprises an initiator object which exchanges messages with one or more dynamically-varying number of receiver objects. The transaction completes at the initiator object. It is assumed that transactions are completed in a finite time and that the initiator object of the transaction is aware of completion of the transaction. Each object port has a structure and a protocol that implements the transaction for that interface. A dependent transaction is a transaction in which the completion of the transaction is dependent upon other transactions. An independent transaction is a transaction that completes on its own without any dependency upon other transactions. Every object port can simultaneously initiate and service multiple independent or dependent transactions.

A transaction can comprise one or more transaction cycles. Transaction phases are assigned to transactions cycles. Each transaction phase is assigned a value. The number of transaction phases is specific to the network-oriented application. A dynamically varying number of object ports can be associated with a transaction phase. An object port tuned to an immediately higher transaction phase can observe and control object ports tuned to an immediately lower transaction phase without participation by object ports tuned to the lower phases. Contents of a transaction can be referred to as a data bag to parallel a mail distribution system in which data is carried in mail bags. The data bag of the lower transaction phase and

control are passed to an immediate higher phase. Once the current highest phase is obtained a data bag of the highest phase transaction and control are returned to the object ports of the immediate lowest phase.

Referring to Fig. 2B, as an example of transaction framework 15, object 30a, object 30b and object 30c are connected to circular communication pathway 36. Objects 30a and 30b are in a lower phase represented by phase 0. Object 30c is in a higher phase represented by phase 1. Object 30c receives data bag 35 and control from object 30a and object 30b. Object 30c can modify data bag 35 and returns data bag 35 to object 30a and object 30b. For example, object 30c can be a tester or management object broker who needs to know the contents of lower phase transactions. The tester can modify the inputs to objects during a testing phase. The management object broker can use the contents of the lower phase transactions to support failure and recovery of transactions.

Interaction framework 16 is a group interaction mechanism where the interaction proceeds in steps to provide a connection or a communication path between two or more object ports. An example interaction framework 16 is described in S. Das et al. "Foundations for Concurrency Among Objects", COMPCON, (Feb. 1991) and P. Bhattacharya et al. "Microkernel for Wireless Information Networks", Kluwer Academic, (Spring 1993) the teachings of which are hereby incorporated by reference into this application for all purposes. Interaction framework 16 supports selection and resolution of shared information of all objects involved in a transaction. Every interface of an object is associated with an interaction port clock and all the transactions between the connected object ports are realized using the object ports and the interaction port clock. Interaction framework 16 can be represented by an interaction network having a circular communication pathway, referred to as a ring as described above. Objects can be connected to the ring through object ports. The above described data bags travel around the ring in one direction between groups of connected objects. Each object port can be tuned to a clock phase associated with the ring. The number of object ports connected to a particular clock phase can be dynamically varying. A clock phase is also called a clock port.

From each object port a clock port receives data and two (asynchronous) control signals: a release (indicating end of the step for that phase) and a trigger (indicating the immediate intentions of other connected object ports for that phase). Once all the ports have released and at least one has triggered, the clock port sends an advance signal as well as the collected data back

to the next clock port present along the ring. The next clock port which receives this signal sends that to all the objet ports tuned to that clock port. If no higher phase is present, the clock port send the advance back to the object ports connected to the same phase.

Every object port set interaction proceeds in steps. Each step begins with the receipt of the advance from the interaction clock, after its receipt of the release and/or trigger signals from the object ports in the previous phase, and ends with the signal of the release/trigger signals of the present phase. The state of the application interaction is distributed in the constituent objects and the clock ports that link them. Any number of clock ports can be created and/or destroyed, according to the requirements of the application. Before destruction, all object ports are detuned. When an object port is used in an application, it is connected to a particular phase and it is referred to as being tuned to that phase of the clock port. Only after successfully tuning to a particular phase, can an object port receive an advance and data bag from the clock port. An object port can only be tuned to a phase after the phase has become inactive since object ports tuned to an active phase have currently received advance and the step in the phase is not complete. If an object port is disconnected from the ring, it is referred to as being detuned from a particular phase of the clock port. Once detuned, an object port will no longer receive the data bag or an advance at that object port.

Figs. 3A and 3B illustrate an example of implementation of interaction framework 16 to form interaction network 39. Interaction network 39 comprises objects 30a-30d placed along ring 36 . Each object 30a-d is identified by an objectID value. Each object port 32a-d is identified by a unique PortID value. Clock ring 36 has a buffer size of 1. Each object port 32a-b is tuned to the clock port 38a (phase 0 or base phase) of the clock ring 36. Similarly, the object ports 32c-d are tuned to clock port 38b (phase 1) of the clock ring 36. All these objects 30a-d share the same interface 34. In this example, Object 30a has ObjectID 1, PortID 500, InterfaceID 9 and is tuned to phase 0. Object 30b has ObjectID 1, PortID 600, InterfaceID 9 and is tuned to phase 0. Object 30c has ObjectID 3, PortID 700, InterfaceID 9 and is tuned to phase 1. Object 30d has ObjectID 4, PortID 800, InterfaceID 9 and is tuned to phase 1.

When clock ring 36 is created, the base phase 0 is automatically created. As shown in Fig. 3B, phase 0 has step 1, step 2, step 3, and so on up to step n. Phase 1 advances for step 1 when ports 32a and 32b tuned to phase 0 release, and at least one of object ports 32a or 32b trigger. Clock ring 36 at phase 0 collects data bag 35 and then sends the collected data bag 35

and the advance to object ports 32c and 32d tuned to phase 1. Object ports 32c and 32d tuned to phase 1 can read and modify received data bag 35. When object ports 32c and 32d finish read/write operations, object ports 32c and 32d release and trigger to phase 1, and again clock ring 36 at phase 1 collects data bag 35 and sends the collected data bag 35 and the advance to all the ports tuned to phase 2. In Figure 3A, since there are no ports tuned to phase 2, or any other higher phase, clock ring 36 sends the collected data bag 35 and the advance along clock ring 36 and all the object ports 32a-32b tuned to the base phase get advance and data bag 35 for the next step of the base phase. Accordingly, as step 1 for phase 0 ends, step 1 for phase 1 begins if there are any object ports tuned to that. Similarly, step 1 for phase 2 begins when step 1 for phase 1 ends. If nobody is tuned to phase 2, then step 2 of the base phase starts. This repeats itself until the entire transaction is complete.

In general, arrival of an advance indicates acknowledgment to each of the object ports that whatever was written in the previous step by all object ports has been delivered to all the recipients in the transaction. The clock ring provides persistence to the data since the data is held until other parties access it. If for a particular object port, an advance is not delivered for the next step, then it indicates that the previous step data has not yet been delivered. The object may be notified by other means about any failures. A buffer size of one is provided by the clock ring for each object port between steps in any phase to minimize the complexity of understanding that is needed to handle faults like object failure, clock failure, external transaction recovery, and internal transaction recovery.

Every object port sends out the release signal indicating its completion of the current step for that phase. This release must be made in a finite amount of time so as to ensure continued progress of transactions. During this step, the object port may have read data collected from a previous step, written data which it wishes to have collected and forwarded (this may include multiple writes), or it may have done nothing and immediately signaled a release. In any event, once the object has signaled a release it may no longer read from nor write to the data bag. Release of an object port tuned to a particular interaction network is an event that is generated by the asynchronous object, and indicates that the state of the object is not any more affected by any communication that goes on in the higher phases, upto receipt of the next advance for that phase from the clock. Release signals the dynamic reconfiguration quiescent, sane, points of the

object, and the object state is stable with respect to that interaction upto the receipt of the next advance for that phase.

The object port trigger indicates its immediate intentions to other object ports tuned to the clock ring at the next higher phase. At least one trigger must be received by the clock ring to go to the next step from an object port expressing the intention in order to model time. This single intention signal is broadcasted to all the object ports as the next immediate advance, at the end of the current step. The clock ring keeps track of how many object ports are tuned to each phase, and not the object ports. As soon as it receives all releases and at least one trigger from those object ports, it collects the individual elements to make a data bag, and then sends the data bag and an advance to all the object ports tuned to the next higher phase. Accordingly, the clock is providing synchronization from the outside. It keeps track of varying number of object ports tuned to different phases and then finally sends one advance and the data bag to the next higher phase. This reduces the complexity of object ports tuned at higher phases since each object port has to poll only one event, the advance arrival, and not all the individual events from each of the object ports tuned to the lower phase and thereby provides tetherless collaboration.

Table 1 illustrates an implementation of source code for creating object ports of an appropriate size and number and registering the object ports with human management framework 19, as described below.

Table 1

```
// Source code of an object showing how it creates ports, and what it does
// upon receipt of an advance - the term used is Port Advance Handler
Port Creation:
        ret = OMF_CreatePort(PortID, PortSize);
// Creates port with unique port id (PortID) which is an integer, and PortSize is in bytes.
        ret = OMF_Register(PortID);
// Port is created by the object and then registered with the management object.
// No connection establishment related code is inside the objects.
// Connection-Oriented connection is established by a third party from outside, and only
// then the first advance is received by the port. The action that is performed by every
// port tuned to a particular phase of the clock is called the Port Advance Handler. The
// following is the template of the Port Advance Handler.
```

```
// Just before receiving the advance, the object state is stable with respect to this interface since
// there are no inputs. Objects must maintain the previous stable state with respect
// to this interface
//Start of Port Advance Handler.
If ret = OMF_Advanced(PortID){
// If the above is true, the port has advanced. The object port has received the advance and the
// collected bag sent by the clock which is in the management object.
// The object state is affected depending on the application information received from
neighboring objects connected to this interface. Change the object state anywhere accordingly//
        Do
        // The following ReadData function returns true if there is one more element to read from
        // the collected bag of the clock.
                If (OMF_ReadData(PortID) == false) then
                        // There are no more elements in the bag to read. Exit the Do loop.
                        Exit Do
                End if
// Then read the element into the variable RecvData. If could be a structure or a string
// depending on the interface type. If you know the type you could specify, else specify an
// array of bytes and read the raw data into that array.
                ret = OMF_LoadDdata(PortID, SizeOf(RecvData), RecvData)
        // Here read all the different messages received and then process them accordingly.
        // If the object needs to perform some actions based on different messages, call
        // such appropriate methods and generate the output.
        Loop  // End of DO READ LOOP.
// Make up the output response to be sent to different messages received, or send out
// requests to other objects for getting services. Then write the data to the port as many times as
// you want. This is application protocol specific. There may not be anything to write.
        ret = OMF_WriteData(PortID, SizeOf(SendData), SendData);
// At this point, say, the object state is stable with respect to this interface. This must be
// reached in finite amount of time. The state can be anything. But that stable state must be
// maintained by the object until the receipt of the next advance at this port. In the meanwhile,
```

```
// the same stable object state may change due to arrival of other ports' advances.
// The application object can give out the release signal indicating the sane point needed for
// dynamic reconfiguration.
        ret = OMF_Release(PortID); // Release signal sent to the port
// The following is the trigger signal indicating immediate intentions to other object ports
// connected to the clock. This is again application specific.
        ret = OMF_Trigger(PortID); // Trigger signal sent to the port
]
// End of Port Advance Handler.
```

Modified CORBA IDL framework 18 is an object-oriented technology in software development which is supported by environment 10. Modified CORBA IDL framework 18 includes the CORBA standard as described in a document entitled "The Common Object Request Broker: Architecture and Specification," published 1992 by the Object Management Group, hereinafter referred to as CORBA specification, the teachings of which are hereby incorporated by reference into this application for all purposes. It will be understood, however, that features of the present invention may be applicable to different implementation of object-oriented systems. The CORBA specification defines an Object Request Broker (ORB) to enable objects to transparently make and receive requests and responses in a distributed environment. Object services are a collection of service implementing interfaces and objects that support basic functions by using and implementing objects, as described in CORBA services: Common Object Service Specification. Common Facilities are a collection of services that applications may share which are less fundamental than object services, as described in CORBA facilities: Common Object Facilities Specification.

Clients request services by issuing requests. A request is an event, such as something that occurs at a particular time. The information associated with a request consists of an operation, a target object, zero or more (actual) parameters, and an optional request context. An object reference is an object name that reliably denotes a particular object. Specifically, an object reference identifies the same object each time the reference is used in a request. An object may be denoted by multiple, distinct object references. A request may have parameters that are used to pass data to the target object; it may also have a request context which provides

additional information about the request. Results and exceptions (if any) may be returned to the clients.

Objects can be created and destroyed. The outcome of object creation is revealed to the client in the form of an object reference that denotes the new object. A type is an identifiable entity with an associated predicate (a single-argument mathematical function with a boolean result) defined over values. A value satisfies a type if the predicate is true for that value. A value that satisfies is called a member of the type. There are two basic data types defined in the CORBA standard: basic types which include integers, floating point numbers, characters, boolean, enumerated, and string; and constructed types which include a record, a discriminated union, a sequence, an array, and an interface.

An interface is a description of a set of possible operations that a client may request of an object. An object satisfies an interface if it can be specified as the target object in each potential request described by the interface. An interface type is a type that is satisfied by any object that satisfies a particular interface. Interfaces and operations are defined in the Modified CORBA IDL. Definitions of the interfaces to objects can be defined in two ways. Interfaces can be defined statically in the modified CORBA IDL. This language defines the types of objects according to the operations that may be performed on them and the parameters to those operations. Alternatively, or in addition, interfaces can be added to a CORBA Interface Repository Service.

As defined in the CORBA Specification, a client sends a request to an object, through the ORB. The client is the entity that wishes to perform an operation on the object, and the Object Implementation is the code and data that actually implements the object. The ORB is responsible for all of the mechanisms required to find the Object Implementation for the request, to prepare the Object Implementation to receive the request, and to communicate the data making up the request. The interface the client sees is completely independent of where the object is located, what programming language it is implemented in or any other aspect which is not reflected in the object's interface.

To make a request, the client can use the CORBA Dynamic Invocation interface or an CORBA IDL stub. The client can also directly interact with the ORB for some functions. The Object Implementation receives a request as an up-call either through the CORBA IDL generated skeleton or through a dynamic skeleton. The Object Implementation may call the

Object Adapter and the ORB while processing a request or at other times. The client performs a request by having access to an Object Reference for an object and knowing the type of the object and the desired operation to be performed.

In the CORBA architecture, the ORB is not required to be implemented as a single component, but rather is defined by its interfaces. Any ORB implementation that provides the appropriate interface is acceptable. The interface is organized into three categories: operations that are the same for all ORB implementations; operations that are specific to particular types of objects; and operations that are specific to particular styles of object implementations. The ORB Interface is the interface that goes directly to the ORB, is the same for all ORBs and does not depend on the object's interface or object adapter.

Modified CORBA IDL framework 18 also includes the following points which describe the implementation used in the present invention. Modified CORBA IDL framework 18 is designed for asynchronous components in which both caller and callee are asynchronous. Every function description is associated with the key work 'async' to indicate the asynchronous behavior. Accordingly, a sender need not be blocked until receipt of acknowledgement from recipient. In addition, since objects interact using interaction framework 16 designers do not need to be concerned about buffer losses.

The modified CORBA IDL framework 18 defines transaction multi-cycle steps. The following is a skeletal specification in the modified CORBA IDL framework 18 of a multi-cycle transaction using interaction framework 16.

```
Interface MYLocalTypeName {
        Attribute INT Message Type;
        ... ... ... ... ... ...
        OMSOFT_TransactionName - "AddSum";
        OMSOFT_DependentIndependent - "Independent";
        OMSOFT_InitiatorReceiver - "Initiator";
ADVANCED_STEP 1:
        //Describe all the relevant methods here for step 1.
        //Perform read
                //Process all the input messages by taking appropriate actions
```

```
        //Write to the port necessary information.
    async OMF_Release(PortID), async OMF_Trigger(PortID);
ADVANCED_STEP 2:
    ... ... ... ... ... ...
    async OMF_Release(PortID), async OMF_Trigger(PortID);

        //Put other steps here
ADVANCED_STEP n:
    ... ... ... ... ... ...
    async OMF_Release(PortID), async OMF_Trigger(PortID);
```

Interface 34 can be implemented using modified CORBA IDL framework 18. System wide unique global handles are provided in objects 30 which include references, ObjectID, PortID, InterfaceID, clock ring 36 and clock phases 38a-b. The global handles are independent of where they are located and what programming language they are implemented in.

Fig. 4 illustrates an implementation of human management framework 19 as a distributed management network 40. Objects 30 and interface 34 (not shown) are managed by management objects 42. Object versions of objects 30 are distributed to management objects 42. Management objects 42 act as centralized management of information about connected objects 30. Management objects 42 send information to human manager object 44 over interface for network evolution (INE) interface 45. Human manager object 44 makes decisions on interface evolution based on information received from management objects 42. Human manager object 44 joins information provided by management objects 42. The complexity at each management object 42 is O(TNLog(N). If there are no management objects 42 then MxT is the total number of objects 30 in the application. The complexity of distributed selection of compatible systems is O((T+M)NLog(N)). Accordingly, if T and N are constants, the complexity of the distributed selection of compatible system is N Log (N). Human manager objects 44 support a graphical user interface (GUI) 46 to allow human manager 47 to interactively manage objects through management object 42 in order to configure and control evolution of the application. Designs for objects 30 are application specific.

Fig. 5 illustrates an implementation of stages of life cycle framework 20. Requirements stage 50 determines constraints such as cost, deadline, reliability or size of the object code.

Project Management planning stage 51 determines major components of software development such as deliverables, milestones and budget to provide a project management plan. Specification and object-oriented analysis stage 52 divides requirements developed from requirements stage 50 into a set of objects 30 having interface 34 to produce a specification. The division of the requirements can be dependent on the project management plan developed in project management planning stage 51 and a desirable level of reusability of objects. The reusability of objects can be determined by reviewing interface 34 and objects 30 for objects which match, either totally or partially. Completely matched objects 30 can be totally reused. Partially matched objects 30 can be assigned to developers in order to extend the existing functionality to meet the new requirements.

Design stage 53 determines objects 30 and interface 34 based on the specification developed in specification and object oriented analysis stage 52. A plurality of developers can determine objects 30 using negotiation between developers developing objects 30 which interact. The developers negotiate structure and protocol for interface 34.

Implementation stage 54 determines implementation of objects 30 and interface 34. Single unit testing can be performed by third parties in automatic single unit testing stage 55. After successful testing, the implementation can be registered with management framework 19, as described below. Compatible systems are determined in detection of compatible systems stage 56.

During a network oriented source code walk through stage 57, tests are performed on the implementation of objects 30 and interface 34. Test results are forwarded to management framework 19 which forwards results to concerned developers and human managers.

Network oriented integration testing stage 58 tests integration of objects 30 to check that objects 30 are correctly combined to provide a product that satisfies the specification determined in specification and object oriented analysis stage 52. Interface input and output of objects 30 can be tested. Test results are forwarded to management framework 19 which forwards results to concerned developers and managers.

Maintenance stage 59 provides support for changes to the software developed in stages 50-58 after acceptance. During maintenance, tested versions of objects 30 can be dynamically updated. Maintenance stage 59 can return to requirements stage 50, project management planning stage 51, specification and object oriented analysis stage 52, design stage 53,

implementation stage 54, network oriented source code walk through stage 57 or network oriented distributed semantic testing stage 58.

Human dimension framework 21 assigns relevant activities to software development team members during life cycle stages as described in life cycle framework 20. An implementation of assignments by human dimension framework 21 is illustrated in Table 2.

Table 2

Stages	Product related	Program related
Requirements 50	Gather cost and deadline, reliability, performance.	Create management network necessary for rapid prototyping. Document requirements.
Project planning 51	Produce milestones, deliverables and budget.	Create software project management plan: delineating separate stages.
Specification and object oriented analysis 52	Create specification document which precisely specifies what the product must do and all constraints that the product must satisfy.	Set up the network of management objects by add/delete Browse repository for reusable objects. Divide core requirements into a set of objects and interfaces as needed. Specify the configuration. Assign ObjectIDs and InterfaceIDs, using distributed name control.
Design 53	Meet with client to clarify any ambiguous requirement.	Tune developers into negotiation processes; Check negotiation, conflict check, add, remove, relocate, and/or replace developers and managers.
Implementation 54 Automatic Testing 55		Developers perform program related implementation and testing.
Detection of compatible application systems 56		Builds the compatible systems.
Network Source Code Walk Through 57		Instruct testers to perform network-oriented source code walk-through of the chosen version of a compatible system.
Distributed semantic testing 58	Test compatible systems on a daily basis	Determine initial conditions of the test. Prepare the project management plan for testing. Add/delete/replace/relocate testers.
Maintenance 59	Generate list of corrective maintenance requests (bugs) received from clients	Set up the product. Dynamically update latest tested versions of objects (consistent and transparent)

In specification and object-oriented analysis stage 52, the developers obtain the entire specification for each object 30 that describes the initial object 30 and interface 34. Thereafter, the developers may be asked to enhance a particular version of an already available object 30 to

meet new requirements or may be given only the InterfaceIDs for object 30 without any structure and protocol for each of the interfaces. In the second case, the developers have to completely design a new object 30.

In design stage 53, the developer can use a negotiation process with other developers. For example the developers can write negotiation scripts in modified CORBA IDL for each interface 34 at each step of the negotiation. The negotiation scripts are registered with human management framework 19. The developers receive negotiation scripts written as modified CORBA IDL descriptions from other connected developers. The developers can use the received information which expresses views of other developers about the structure and protocol for that particular interface to revise the negotiation scripts. After completing negotiation of all the interfaces, the developers write an object interface description in modified CORBA IDL and registers the object interface description with management framework 19. Management framework 19 returns code generated from the modified CORBA IDL description to the developer. An implementation of a preferred negotiation process is described in greater detail below.

In implementation stage 54, the developers complete implementation, and perform single-unit testing. The developers register implemented objects 30 with management framework 19, for example object versions can be increasing as a result of modifications after testing. The detection of compatible application systems 56 builds compatible application systems. The developers take necessary actions based on the results of network-oriented source code walk-through stage 57 and distributed semantic testing stage 58.

In the network oriented integration testing stage 58 distributed semantic testing is performed to check that objects 30 are correctly combined in order to achieve a product that satisfies its specifications. During network oriented integration testing stage 58 interface 34 are carefully tested. Human testers can be used to test interfaces, states, and input/output to the objects. The testers identify any faults, categorize them and enter the results to management framework 19 without fixing the faults.

During maintenance stage 59, the maintenance managers notify the maintenance programmers by delivering a fault report, or specifications to enhance existing objects 30. Maintenance programmers provide corrective and adaptive maintenance changes and act as a

combination of testers and developers in order to find and fix bugs in the software developed using stages 50 – 59 described above.

The following is an example of development of a network application in distributed in software development environment 10 using lifecycle framework 20. During requirement stage 50, information about the network application is gathered from an end user or a client. Fig. 6 illustrates network application 60 generated by requirements stage 50. Network application 60 comprises object 30a, having ObjectID 01, hereinafter referred to as Object 01, object 30b having ObjectID 02, hereinafter referred to as Object 02, and object 30c having ObjectID 03, hereinafter referred to as Object 03. Each object 30a-c has a respective state object port 61a-c through which a state of each object 30 can be set or received. Each object 30a-30c has a state interface 63a-63c assigned respectively as state InterfaceIDs 1, 2, 3. State InterfaceIDs 1, 2, 3 are assigned respectively to Object 01, Object 02 and Object 03. Each object 30a-c has a respective basic service object port 62a-c through which each object 30 sends out a request or receives request, processes request, and sends back response through basic service object port 62 can be object ports 32, as described above. Basic service interface 64 connects basic service object port 62 of each of Object 01, Object 02 and Object 03. Interface 64 is assigned InterfaceID 4. It will be appreciated that state object ports 61a-c and basic service operation ports 62a-c can comprise object port 32 and state interface 63 and basic service interface 64 comprises interface 34.

In network application 60, Object 01 sends a request message over basic service interface 64 to Object 02 to perform a square operation. Object 01 receives from Object 02 a squared number via basic service interface 64. Object 01 sends a request message to Object 03 to perform a summation operation over basic service interface 64. Summation(n) is performed by adding n, n-1, ..., 1, resulting in n(n+1)/2. Object 01 receives the summed number over basic service interface 64 from Object 03. It is assumed that integer numbers are passed between Object 01, Object 02 and Object 03. Accordingly, Object 02 and Object 03 only provide services and do not generate any requests.

In specification and object oriented analysis stage 52 interface IDs 1 through 4 can be set to null. The following specification can be determined. Object 01 has two interfaces having Interface IDs 1, 4. Interface ID 1 is an interface for state port 61a which supports get and set states and also supports destroying Object 01 by a set state operation. Basic service object port

62a can send out a square or a summation request operation and receive back the appropriate response. Object 02 has two interfaces having Interface IDs 2, 4. Interface ID 2 is an interface for state port 61b which supports get and set states and also supports destroying Object 02 by a set state operation. Basic service object port 62b receives the square request, computes and outputs the response. Object 03 has two interfaces having Interface IDs 3,4. Interface ID 3 is an interface for state port 61c which support get and set states, and also support destroying Object 03 by a set state operation. Basic service object port 62c of Object 03 receives the summation request, computes and outputs the response. In design stage 63, a manager assigns the task of designing Object 01, Object 02, and Object 03 to three developers identified as D1, D2, and D3 respectively.

Environment 10 can be used to implement negotiation between developers, during software development of an application in design stage 53. Fig. 7 is a flow diagram of an implementation of a method of tetherless negotiation 70 in accordance with the teachings of the present invention. Tetherless negotiation provides negotiation by proceeding in steps. In block 72, human manager object 44 assigns the task of designing and implementing objects to developers. In block 73, a negotiation network is established between developers and management objects 42. Fig. 8 is an implementation of negotiation network 80 which can be used for a similar application network 60, shown in Fig. 6. Negotiation network 80 comprises developers D_1, D_2, D_3. Developers D_1, D_2, and D_3 are respectively assigned to develop objects 30 referred to as Object 01, Object 02 and Object 03 by human manager object 44. In block 73, human manager object 44 can specify parameters in establishing the negotiation network 80. For example, human manager object 44 can specify in establishing negotiation network 80 the login name, password for each of developers D_1, D_2, and D_3, and Object ID, Interface2D for objects 0_1, 0_2, and 0_3. Each developer D_1, D_2 and D_3 creates a respective developer negotiation object port 82a, 82b and 82c for object 30a, object 30b and object 30c. Each developer D_1, D_2 and D_3 creates respective object ports 32a, 32b, 32c as developer state ports. Developer negotiation and state ports are registered with management object 42. Developers use their state object port to gel design specification, object ID, InterfaceID, and other services that the developers need as part of life cycle framework 20. At the end of implementation and single unit testing stage 54, the developers check in the objects with the management object 42 with their implementation. The management object 42 uses the same interface to return object version information to the developers. Each developer uses this interface to check in the modified IDL at the

end of each step of the negotiation process. Human manager object 44 creates a corresponding management side state ports object 84a-84c in management object 42 and INE management object port 83. Human manager negotiation object port 85b and human manager INE port 85a are established in human manager object 44.

Human manager INE port 85a, human manager negotiation port 85b, management INE port 83, management negotiation ports 84a-c are registered with management object 42. Human manager object 44 creates a clock and connects human manager INE port 85a and management INE port 83 on the management side. Using INE interface 45, human manager 47 (not shown), using human manager object 44, controls the entire negotiation process. Human manager object 44 sends the negotiation relation data to management object 42. Management object 42 enables connections between developer state object ports 32a-c and the management negotiation ports 84a-c on the management object 42. In this scenario, network 80, human manager 47 decides to view the negotiation process directly. Negotiation clock 86 is created in management object 42. Human manager negotiation object port 85b of human manager object 44 tunes to the base phase of negotiation clock 86. When port 85b gets the advance, it is held. All three developer negotiation object ports 82a-c are tuned to a high phase of negotiation clock 86. Port 85b then releases the port and the developer negotiation object ports 82a-c get the advance signal.

In block 77, developers negotiate with one another until all developers agree on the negotiation. For example, developers D_1, D_2 and D_3 can iteratively agree by forwarding through developer negotiation object ports 82a-c negotiation scripts written in modified CORBA IDL framework 18 and English with advance, release and trigger mechanisms described above. The modified CORBA IDL written by the developers at the end of each step of the negotiation are checked for syntactic correctness by the backend IDL compiler supported in the developer's environment.

In this example, developers D_1, D_2, and D_3 negotiate to agree upon the structure and protocol for interface version 1.0 interface which is a common global interface InterfaceID 4. Any developer D_1, D_2, D_3 can negotiate the complete description of the structure and protocol for the InterfaceID = 4 that is consistent with their individual designs. For example, in step 1, developers D2 and D3 may not want to propose anything. Accordingly, D2 and D3 release and may or may not trigger step 1. In step 1, developer D1 makes a proposal. Table 3 is an example of a negotiation script for developer D1.

Table 3

Negotiation script written by Developer D1 for Step 1, InterfaceID = 4.

```
// Start of the Negotiation Script for Step 1 of Object 01, for InterfaceID = 4. Developer D1.
// For InterfaceID 4, Interface Version = 1.0 I would like to have the structure:
structure InterfaceID4 {
        int MessageType;
        int Data;
};
Independent transaction name: TRANSAC_NAME_SQUARESUM;
Initiator will be 01; one or more receivers; must end in a finite time at the initiator.
message types:      #define OM_SQUARE              1
                    #define OM_SQUARE_OUTPUT       2
                    #define OM_SUMMATION           3
                    #define OM_SUMMATION_OUTPUT    4
Protocol: D1's object 01 will send message type OM_SQUARE, and put the number which
has to be squared in the data field of the structure. 01 will receive message type
OM_SQUARE_OUTPUT and the squared output from the object 02, in the data field of the
same structure. D1's object 01 will send message type OM_SUMMATION, and put the
number which has to be summed in the data field of the structure. 01 will receive message
type OM_SUMMATION_OUTPUT and the summed output from the object 03, in the data
field of the same structure. Both are part of the same port and interface of the object 01.
// End of the Negoitation Script.
```

Developer D1 forwards the negotiation script to developer negotiation object port 82a and then releases and triggers developer negotiation object port 82a. Trigger is sent to exchange developer D1's negotiation script at the earliest possible time to developer D2 and developer D3. At this point of time, developers D1, D2 and D3 have released and triggered, and only developer D1 has written to developer negotiation object port 82a. Negotiation clock 86 collects the negotiation script and makes up data bag 35 (not shown). In this case, only one negotiation structure element is in data bag 35, since only developer D1 forwarded a negotiation script in the current step. Negotiation clock 86 delivers data bag 35, and the advance to human manager negotiation object port 85b which is tuned to a different phase 0. Upon receipt of the advance, human manager object 44 can look into the data, the

received negotiation script, and understand the proceedings of the group negotiation step 1. Human manager object 44 understands that developer D1 has proposed a change. After completing inspection, human manager object 44 writes the data back to human manager negotiation object port 85b, so that the elements received by human manager negotiation object port 85b are not destroyed or modified, and can be delivered to developers D2 and D3 which are connected to a different phase 1. Human manager object 44 releases and triggers human manager negotiation object port 85b for the base phase. Negotiation clock 86 collects this new phase data and makes up data bag 35. Negotiation clock 86 delivers data bag 35, and the advance to developer negotiation object ports 82a-c which are tuned to phase 1. Upon receipt of the advance, developer D1 may not do anything in this step 2. Developer D1 releases and may or may not trigger. Developer D2 and developer D3 read and understand the proposed change from developer D1 and can disagree or completely agree upon the individual portions of the structure and protocol. If both developer D2 and developer D3 agree then the negotiation is completed. Alternatively, the negotiation is iterated until all developers D1, D2 and D3 agree. Table 4 is an example of a negotiation script of developers for step 2.

Table 4

Negotiation script of developer D2

```
//Start of negotiation script step 2, developer D2
//Response to D1's request
//Structure and protocol fine with me. I will implement OM_SQUARE and
//OM_SQUARE_OUTPUT with the corresponding numbers. Usage of data field is OK with
me.
//End of negotiation script step 2, Developer D2.
```

Table 5 is an example of a negotiation script of Developer D3 from step 2.

Table 5

Negotiation script of Developer D3

```
//Start of negotiation script step 2, developer D3 – Response to D1's request
//Structure, protocol fine with me, will implement OM_SUMMATION and
//OM_SUMMATION_OUTPUT with the corresponding numbers. Usage of data field is OK.
//End of negotiation script step 2, Developer D3.
```

Both developer D2 and developer D3 write their individual acceptance statements in modified COBRA IDL and English to their respective developer negotiation object ports 82b and 82c and

release and trigger respective developer negotiation object ports 82b and 82c. Negotiation clock 86 again delivers a collected data bag 35 and advance to human manager object 44. Human manager object 44 sees that developer D2 and developer D3 have agreed. Accordingly, human manager object 44 does not intervene to resolve any conflicts in this step. Human manager object 44 writes the received elements without any change back to human manager negotiation object port 85b and sends release and trigger to human manager negotiation object port 85b. Negotiation clock 86 delivers this new data bag 35 and the advance to developer object ports 32a-32c. Developer D2 and developer D3 may not do anything. Developer D1 understands that developer D2 and developer D3 have agreed and that is the structure and protocol for the InterfaceID = 4, for Interface Version = 1.0.

In block 78, developers D1, D2 and D3 complete implementation in implementation stage 54 of their individual object versions that include the InterfaceID = 4, InterfaceVersion = 1.0. Each of developers D1, D2 and D3 write implementation_completed message and release/trigger their negotiation ports. When human manager negotiation object port 85b gets the advance and looks into the contents, human manager object 44 understands that the developers have completed implementation. Human manager object 44 writes, releases/triggers human manager negotiation object port 85b. Object ports 32a-32c get advance for the fourth step. All developers D1, D2 and D3 see that the implementation is complete and at this point of time, negotiation transaction for InterfaceID = 4, InterfaceVersion = 1.0 is complete. At this time, new developer object ports 30 can be added to this negotiation process and a new transaction can be started by any developer in the newer configuration.

As described above, human manager object 44 tunes to every negotiation proceedings in the application, and transparently observes and controls the negotiation processes. In large network applications, there may be multiple hundreds of active different negotiation groups. Human manager object 44 may need hundreds of ports to observe and control, from outside. Instead of human manager object 44 directly tuning to all the negotiation proceedings (by creating ports in the manager object), human manager object 44 can use management objects 42 having management negotiation object ports which are already tuned to all the clocks in the application, to get all the information (negotiation proceedings). This approach involves human manager object 44 sending a number of commands to management objects 42, but reduces the number of ports in human manager object 44, wherein the original approach reduces the number of request messages between human manager object 44 and management object 42. The approach used can be chosen based on the network application.

In general, since the advance is sent by the negotiation clock 86, and not the developers themselves, the attention of the developers is diverted away from the primary task of design, only at the start of each new step (with the arrival of advance), independent of the factors mentioned before, accordingly is tetherless negotiation (collaboration). In the above described negotiation, human manager object's 42 involvement at a different phase is totally transparent to the developers. When the human manager object gets an advance at the base phase, by then all the higher phase developer negotiation ports are already voluntarily released, and developers do not know where the data is going. The human manager's object port gets one advance and the collected bag for the entire group of developers at each step, thereby improving the human manager's productivity and decreasing complexity. Also, the human developers do not know how many developers are connected to the negotiation process, where they are located, and when they work. This can improve the developers' productivity. Each developer waits for an advance to come at the negotiation port for the next step, which is independent of the factors mentioned before. The number of elements in the bag, that is received along with the advance, indicates how many developers are involved and it is dynamic, and is set by the clock. The arrival of advance to all the ports tuned to a different phase is controlled by the clock and also depends on how many developers are tuned to the clock and varies dynamically.

Table 6 are designs at the end of negotiation which is frozen in the modified CORBA IDL.

Table 6

Step 1:when the object port advances for step 1, D1's Object 01 writes the appropriate message to the basic service object port thereby initiating the transaction. It could be either the perform square request or the perform summation request depending upon the user's choice. The data entered by the user is copied onto the structure with the appropriate message type. Then Object 01 releases and triggers that basic service object port for step 1. When Object 02 and Object 03 receive step 1 advance, they release and trigger.

Step 2:when object port advances for step 2, Object 01 does nothing other than release and trigger. Object 02 and Object 03 read the data, interpret it. If it is a square request, Object 02 computes the square, copies the response onto the output structure with the appropriate message type. Object 02 writes the output structure to the basic service object port, releases and triggers the port. Object 03 does nothing other than release and trigger. If it is summation request,

Object 03 computes the response and writes it to basic service object port. Object 03 and Object 02 release and trigger their basic service object ports.

Step 3:when object port advances for step 3, Object 01 reads the incoming data, interprets it and understands that it has received the final output. This indicates end of the transaction at the initiator (Object 01). Object 02 and Object 03 release and trigger their basic service object ports without writing anything. Object 01 can start another transaction cycle immediately or just release and trigger without writing anything to the port.

In implementation stage 54, a developer can follow the steps shown in Fig. 9 for implementing network application 60, shown in Fig. 6. In block 90, states of objects 30a-c are determined and respective state interfaces 63a-c are implemented for supporting objects 30a-c get and set states and destroying of objects 30a-c. For example, states can be chosen as a corresponding integer number which reflects upon a transaction in network application 60. In network application 60, each of state ports 61a-61c are defined to have two fields: a first field is a message type field indicating set or get operation; and a second field is a statevalue field containing the state of objects 30a-c. Developer D1 chooses Object 01's initial state to be 0. Accordingly, whenever Object 01 sends out a request for a service, its state changes to 1. Upon receipt of a response, Object 01's state is changed to 0. Similarly developer D2 and developer D3 initially set their corresponding object states of Object 02 and Object 03 to 0. When Object 02 or Object 03 receive a request from Object 01, the state is changed to 1. As soon as the response is forwarded in the same step, the state is changed back to 0. For example, state message types can be assigned the following definitions: 1 indicates set, 2 indicates get, 3 indicates set acknowledgement (ack), 4 indicates get acknowledge, (get_ack), 5 indicates, requested to die; 6 indicates received and now dying. In accordance with the teachings of the present invention these message numbers and the state type could vary between objects and it is decided by the developers, but the overall support must be provided in each object by the corresponding developers.

In block 92, an object port is associated with each object interface of the objects. Accordingly, state ports 61a-61c are associated with respective state interfaces 63a-63c and basic service object ports 62a-62c are associated with basic service interface 64. In block 94, transactions are determined for processing request and incoming messages. In block 96, interfaces can be described using modified CORBA IDL. A separate modified CORBA IDL

description is determined for each object interface. The modified CORBA IDL description describes structure and transactions of object interfaces including, for example, Interface ID, Interface Version, and Initiator/Receiver Information. In network application 60, developer D1 writes two separate modified CORBA IDLs in Object 01 for each of the two interfaces having Interface ID 1 and 4. An implementation of the modified CORBA IDL description for Object 1 Interface ID 1 is shown in Table 7.

Table 7

Modified CORBA IDL description for the InterfaceID = 1 of object 01 written by D1.

```
//Modified CORBA IDL description for Object 01; written by developer D1 //
//InterfaceID = 1, InterfaceVersion 1.0 STATE PORT of 01 //
interface StateObject1 {
// Includes some more methods for set and get operation. File: 01_Interface1.idl
// StateObject1 is the LocalTypeName inside 01 for state interface.
// MyLocalState is the local state variable holding current state within the object.
// MessageType = 1 is a SET operation. Object state can be set from outside.
//              MyLocalState = InputStructReceived.StateValue;
// MessageType = 2 is a GET operation. Object internal state is needed outside.
              OutputStructToBeSent.StateValue = MyLocalState;
// MessageType = 3 is SET_ACK to the set operation. Object sends SET_ACK to
              the external object upon receiving the SET request.
// MessageType = 4 is SET_ACK to the set operation. Object sends GET_ACK to
              the external object upon receiving the GET request.
// MessageType = 5 is REQ_TO_DIE operation. Object must die.
// MessageType = 6 is RECEIVED_REQ_TO_DIE is the ACK sent by the object
              in response to message 5. In the next step, when the object
              ensures that the other party has received message6, object exits.
struct x {Attribute int MessageType;
        Attribute int StateValue;
struct x in, out;
        if OMF_Advanced (PortID)
        {       if (In.MessageType = = 1)
```

```
                mylocal state = in. stateValue;
                out.MessageType = 3;
            end if

            if (In.MessageType = = 2)
                out.StateValue = mylocal state;
                out.MessageType = 4;
            end if

            if (In.MessageType = = 5)
                out.MessageType = 6;
                cleanup( );
            end if
        OMF_WRITE(PortID, OUT)size of(struct out))
        OMF_Release (PortID);
        OMF_Trigger (PortID);
        };
}
```

An implementation of the modified CORBA IDL description for Object 01, Interface ID 4 is shown in Table 8.

Table 8

Modified CORBA IDL description for the Interface ID = 4 of Object 01 written by D1.

```
//Modifed CORBA IDL description for Object 01; written by developer D1 //
// InterfaceID = 4, InterfaceVersion 1.0 BasicServicePort of 01 //
// Includes some more methods. File: 01_Interface4.idl
// MyLocalState is the local state variable holding current state within the object.
interface BasicServicePortObject1 {
        attribute int MessageType;
        attribute int Data;
```

// PortID is the Unique Service ID of the object port in the entire application system.
// BasicServicePortObject1 is the LocalTypeName inside 01 for InterfaceID=4
//Independent transaction is: TRANSAC_NAME_SQUARESUM
// Initiator of the transaction and must end at 01
Step 1:
Writes to the Port MessageType: OM_SQUARE (1) or OM_SUMMATION (3)
// Writes to the Port Data: USer Input Data
MyLocalState = 1;
// Stable object state is reached; now release the port
// OMF_Release(PortID); OMF_Trigger(PortID);
Step 2:
// Object state is stable (no change in state since there are no valid inputs)
OMF_Release(PortID); OMF_Trigger(PortID);
Step 3:
// Read from the Port MessageType: OM_SQUARE_OUTPUT (2) OR
OM_SUMMATION_OUTPUT (4)
// Read Data from the Port, display it, End of the transaction at the initiator in step 3.
MyLocalState = 0;
// If there is a valid data from the user, write again by starting the next transaction
// 01 could have held the control. But writes nothing in this step.
OMF_Release(PortID); OMF_Trigger(PortID);
};

An implementation of the modified CORBA IDL description for Object 02, Interface ID 4 is shown in Table 9. Developer D2's IDL description for the InterfaceID = 2, and developer D3's modified CORBA IDL descriptions can be similarly determined.

Table 9

Modified CORBA IDL description for the InterfaceID = 4 of object 02 written by D2.

//Modified CORBA IDL description for Object 02; written by developer D2 //
// InterfaceID = 4, InterfaceVersion 1.0 BasicServicePort of 02 //
// Includes some more methods. File: 02_Interface4.idl
// MyLocalState is the local state variable holding current state within the object.

```
interface BasicServicePortObject2 {
        attribute int MessageType;
        attribute int Data;

// PortID is the Unique Service ID of the object port in the entire application system.
// BasicServicePortObject2 is the LocalTypeName inside 02 for InterfaceID = 4
// Independent transaction is:  TRANSAC_NAME_SQUARESUM
// Receiver of the transaction
Step 1:
        OMF_Release(PortID); OMF_Trigger(PortID);
Step 2:
        Read from the Port MessageType:  OM_SQUARE (1)
        MyLocalState = 1;
        // Read from the Port Data:
        Response = ComputeSquare(InputStructRecd.Data)
        // Write to the Port Messagetype:  OM_SQUARE_INPUT (2)
        Write to the Port Data:  Response is written to the Data field.
        MyLocalState = 0;
        // Object state is stable
        OMF_Release(PortID); OMF_Trigger(PortID);
Step 3:
        // End of the transaction in step 3 at the initiator
        OMF_Release(PortID0: OMF_Trigger(PortID);
};
```

In block 97, each determined object interface is implemented as a separate source file. The separate source files determine the overall object code for network application 60. For example, source code for Object 01 includes two ports; state object port 61a having StatePortID and basic service port 62a having ServicePortID of the appropriate sizes. The PortIDs indicate that the two port services are different and are uniquely identified in the application system. In order to access the service provided by those ports, the human manager must know the corresponding PortIDs. PortIDs or ServiceIDs are

similar to the capabilities of the object, which the human manager must obtain from the object in order to utilize its services.

Examples of functions used to create the source code and their descriptions are as follows:

OMF_CreatePort(PortID, PortSize): This method lets objects 30 create a port of size PortSize. Object 30 receives and/or sends data through the object port. PortID is an integer that identifies the object service uniquely in the entire application system. For example, the object port can be object port 32, state object port 61 or basic service object port 62.

OMF_Registration (PortID): This method lets object 30 register with the management its port with PortID to allow the manager to access the object's service.

OMF_Advanced (PortID): This method lets object 30 check whether the object port with PortID has advanced. Returns true if advanced, else false.

OMF_ReadData (PortID): This method lets object ports with PortID to check if there is one more element in the bag to read. Returns non-zero if true.

OMF_LoadData (PortID, PortSize, InputStructure): This method enables loading PortSize bytes from the object port with PortID to the structure InputStructure.

OMF_WriteData (PortID, PortSize, OutputStructure): This method enables writing PortSize bytes from the OutputStructure to the object port with PortID.

OMF_Release (PortID): This method enable releasing the port with PortID. Object port handlers use this to send out stable states of objects when requested by the human manager (through the management object).

OMF_Trigger (PortID): This method enables triggering the object port with PortID.

An example source code developed as an implementation of Object 01 is shown in Table 10.

Table 10

Source code for Object 01 implemented by developer D1.

```
// START OF SOURCE CODE FOR OBJECT 01 by developer D1
// (InterfaceID = 1, InterfaceVersion = 1.0), (InterfaceID = 4, InterfaceVersion = 1.0)
#define OM+SQUARE                1
#define OM_SQUAEW_OUTPUT         2
#define OM_SUMMATION             3
#define OM_SUMMATION_OUTPUT      4
struct StateObject1 {
```

```
        int MessageType;
        int StateValue;
} StateInputStruct, StateOutputStruct;
struct BasicServicePortObject1 {
        int Messagetype;
        int Date;
} ServiceInputStruct, ServiceOutputStruct;
main (int argc, char **argv) {
        int MyLocalState, ServicePortID, MyStateWrite, MyServiceWrite;
        int StatePortID = atoi (argv[1]);
        OMF_CrreatePort(StatePortID, (size of) struct StateObject1);
        OMF_Registration(StatePortID);
        MyStateWrite = 0;
        MyServiceWrite = 0
        ServicePortID = StatePortID + 1;
        OMF_CreatePort(ServicePortID, (size of) struct BasicServicePortObject1);
        OMF_Registration(ServicePortID);
        // Object creates two ports associated with different services
        // Objects register ports with the management so that others could access the
        // object's services or capabilities knowing that number.
        While (1) {
                if (OMF_advanced(StatePortID)) {
                        Do {
                                OMF_LoadData(StatePortID, (size of) struct StateInputStruct,
                                                                StateInputStruct);
                if (StateInputStruct.MessageType == 1({ /* state set */
                        MyLocalState = StateInputStruct.StateValue;
                        MyStateWrite = 3;
                }
                if (StateInputStruct.MessageType == 2) /* state get */
                        MyStateWrite = 4;
```

```
        if (StateInputStruct.MessageType == 5)      /* die received */
                MyStateWrite = 6;                   /* send die ack */
        if (StateInputStruct.Messagetype == 6
                /* confirm the other received die ack */
                exit();
} while OMF_ReadData(StatePortID);
if (M7StateWrite == 3) {      /* set ack */
StateOutputStruct.MessageType = 3;
MyStateWrite = 0;
OMF_WriteData(StatePortID, (size of) struct StateOutputStruct,
                                        StateOutputStruct;
}
if (MyStateWrite == 4)        {       /* get ack */
StateOutputStruct.MesageType = 4;
StateOutputStruct.StateValue = MyLocalState;
MyStateWrite = 0;
OMF_WriteData(StatePortID, (size of) struct StateOutputStruct,
                                        StateOutputStruct);
}
if (MyStateWrite == 6)        {       /* die ack */
StateOutputStruct.Messagetype = 6;
MyStateWrite = 0;
OMF_WriteData(StatePortID, (size of) struct StateOutputStruct,
                                        StateOutputStruct);
}
OMF_Release(StatePortID);
OMF_Trigger(StatePortID);
} /* end of IF_ADVANCED for the StatePortID */

if (OMF_Advanced(ServicePortID)) {
do {
```

```
OMF_LoadData(ServicePortID, (size of) struct ServiceInputStruct,
                                        ServiceInputStruct);
If ((ServiceInputStruct.MessageType == OM_SQUARE_OUTPUT) ||
    (ServiceInputStruct.Messagetype == OM_SUMMATION_OUTPUT)) {
    // END OF THE INDEPENDENT TRANSACTION AT THE INITIATOR
    MyLocalState = 0;
    DisplayMessage(ServiceInputStruct.Data); /* Display the response */
}
} while OMF_ReadData(ServicePortID);
if (Userinput) {
OMF_WriteData(ServicePortID, (size of) struct ServiceOutputStruct,
                                        ServiceOutputStruct);
// START THE TRANSACTION
UserInput = 0;
}
OMF_Release(ServicePortID);
OMF_Trigger(ServicePortID);
} /* end of IF_ADVANCED for the SERVICEPORTID */
//* PrompUser for the operation type after displaying response and the data value *//
//* Then Set the ServiceOutputStruct Datastructure with proper values *//
if (USER_WANTS_TOSEND_SQUARE){
    ServiceOutputStruct.MessageType = OM_SQUARE
    ServiceOutputStruct.Data = InputData from the user;
    //* Set Userinput flag to 1 *//
    Userinput = 1;
} else {
    ServiceOutputStruct.Messagetype = OM_SUMMATION
    ServiceOutputStruct.Data = InputData from the user;
    //* Set Userinput flag to 1 *//
    UserInput = 1;
}
```

} /* infinite while loop */

} /* end of main */

// END OF SOURCE CODE for object 01 //

In block 98, each implemented object 30 is registered with management framework 19 which returns an object version number. Testing of objects can be performed as specified in network oriented source code walk through stage 57 and network oriented integration testing stage 58. Then the developer registers the object with management framework 19. For example, developer D1 can enter the information shown in Table 11.

Table 11

ObjectID = 01

InterfaceID	1	4
InterfaceVer	1.0	1.0
IDL Description	01_Interface1.idl	01_Interface4.idl
SourceCodeFileName	01_Interface1.bas	01_Interface4.bas
LocalTypeName	StateObject1	BasicServicePortObject1

Management framework 19 returns the appropriate object version number. In this case, it returns version 1.0 to developer D1. Similarly developers D2 and D3 register their individual object's information. For both of them, the management object returns 1.0 as the object version number.

Objects and interfaces can evolve during development of network application. During development versions of objects can become incompatible for the following reasons: interface versions may not match; registration of objects 30 with management framework 19 can occur at different times; or objects 30 of older versions are deleted. Fig. 10 is a flow diagram of a method for determining compatible applications 100 in environment 10. In block 101 an object table is determined for combining object versions and interface versions. The column names are the Object ID; Object Version and Interface Ids (for example 1, 2, 3, 4). Values associated with the columns are the interface versions associated with that particular interface. ObjectVersion column indicates object versions. In a preferred embodiment, objects 30 are inserted into the object table in such a way that m implementations remain together as a single block, with only increasing object version numbers, thereby locating all m^{-th} implementations for every unique combination of interface versions $<I_1 \ldots I_m>$.

In a preferred embodiment, object tables are created which are increasing. Accordingly, for every unique version of interfaces the object version keeps increasing. Accordingly, as any fixes and newer versions of interfaces are created they are inserted in the object table to maintain the table as increasing.

All of the object tables are sorted with the number of object tables represented by T. The number of interfaces in the application is represented by I. Object 30 can have multiple interfaces with the maximum number of interfaces an object has represented by Oimax. It is assumed that all interfaces 34, 1-4, 64, and 63a-c in the network application 60 evolve. No interface remains unchanged for a long period of time (when compared to other interfaces in the application), as the application starts evolving. This assumption is used in order to restrict the number of records, in the intermediate stages of a join operation of, the data table selection method, described below, wherein N is the varying number of object versions in the object table. The complexity of distributed selection method is N log N.

Table 12 illustrates an implementation of an object table for application network 60, shown in Fig. 6. For example, Object 01's version 1.0 implements interfaces 1 and 4 and with each interface having interface versions 1.0.

Table 12

Relations obtained for the evolving application.

Table 1:	ObjectID	ObjectVer	Interface Ver ID1	Interface Ver ID4
	01	1.0	1.0	1.0
	01	1.1	1.0	1.1
Table 2:	ObjectID	ObjectVer	Interface Ver ID2	Interface Ver ID4
	02	1.0	1.0	1.0
	02	1.1	1.0	1.0
	02	1.2	1.0	1.1
Table 3:	ObjectID	ObjectVer	Interface Ver ID3	Interface Ver ID4
	03	1.0	1.0	1.0
	03	1.1	1.0	1.1

For example Object 1 includes two object versions 1.0 and 1.1 as shown in Table 12. If a newer version is created for InterfaceID 4, interface Ver 1.0 and InterfaceID 1, interface Version 1.0 it is inserted between object Ver 1.0 and object Ver 1.1. The new object version will be

assigned an object version between Object Ver 1.0 and Object Ver 1.1, for example Object Ver 1.01.

In block 102, the entire table is sorted with respect to the object version. If there are new implementations available for every unique combination of interface version, then the most recent version of the combination is used in the sorted table. For example, Object 2 has object versions 1.0 and 1.1 with the same combination for the interface versions, 2 and 4, as shown in Table 10. Object version 1.1 is the most recent version for this combination of interface versions, m^{th} version. Object version 1.1 is determined after bug-fixing without changing the interface version. The newer object version 1.1 is inserted in the sorted table in order to keep multiple implementations as joint blocks.

Table 13

Object 02 after sort for Object Version

ObjectID	ObjectVer	2	4
02	1.1	1.0	1.0
02	1.2	1.0	1.1

In block 103, the first two object tables are sorted with respect to common interface Ids. For example, Object 01 and Object 02 have a common interface ID 4.

Table 14 shows sorting with respect to Interface ID 4 for Object 01 and for Object 02.

Table 14

Sorting Object 01 and Object 02 with respect to Interface ID 4

Table 1:	ObjectID	ObjectVer	1	4	Table 2:	ObjectID	ObjectVer	2	4
	01	1.0	1.0	1.0		02	1.1	1.0	1.0
	01	1.1	1.0	1.1		02	1.2	1.0	1.1

In block 104, the first two object tables are joined with respect to all common basic service interface 64 of application 60 to form an intermediate join result as shown in Table 15. If there are multiple common interfaces between object tables, a tuple of object versions determined by an AND condition is outputted if all the corresponding interface versions of the common interfaces match.

Table 15

Intermediate output after joining sorted Table 1 and Table 2.

ObjectID	ObjectVer	InterfaceID(4)	ObjectID	ObjectVer	1	2
01	1.0	1.0	02	1.1	1.0	1.0
01	1.1	1.1	02	1.2	1.0	1.0

In block 105, the object table of the next object is sorted with respect to a common interface ID to the immediately prior result. Object 03 shares one common interface ID 4 with Object 01 and Object 02. Table 16 illustrates output after sorting with respect to Object 03 interface ID 4. Block 103-105 can be repeated to process all object tables.

Table 16

After sorting Object 03 with respect to InterfaceID = 4.

Table 3:	ObjectID	ObjectVer	3	4
	03	1.0	1.0	1.0
	03	1.1	1.0	1.1

In block 106, the final join is determined by joining the intermediate result with the sorted next object determined from the previous step. Table 17 shows the final join of Object 01, 02 and 03.

Table 17

Final join output (all the partial order compatible systems).

ObjectID	ObjectVer	ObjectID	ObjectVer	ObjectID	ObjectVer	1	2	3	4
01	1.0	02	1.1	03	1.0	1.0	1.0	1.0	1.0
01	1.1	02	1.2	03	1.1	1.0	1.0	1.0	1.1

The complexity of the above example is NLOG(N)+2N.

Fig. 11A-11B illustrate the evolution of the above described example. Rows with square boxes indicate that the corresponding object is evolving because of bug fixing or changing interface versions. Rows with circles on the right indicate that the corresponding interface is evolving to meet changing application requirements. The connecting edge indicates the interface versions implemented by that object version. Compatible systems are represented by CS-I, CS-II and CS-III, in Fig. 11B. The corresponding interface versions are indicated on the right. The first compatible system is CS-I. Bug fixing in Object 02 version 1.0 results in a new Object 02 version 1.1 and second compatible system CS-II. Changing requirements in interface ID4 results in changing Interface ID 4 Interface version 1.0 to Interface Version 1.1 and third compatible system CS-III.

Fig. 12 illustrates an implementation of block 103 by human management framework 19 in order to obtain shared interface information from Object 01, Object 02 and Object 03. Each of objects Object 01, Object 02 and Object 03 which can be distributed over a plurality of management objects 42, referred to respectively as M1, M2 and M3. For each object, Object 01, Object 02 and Object 03, human management object 42 assigns increasing object version numbers. For example, each object

table as shown in Table 15 can be managed under a separate management object 42. The number of tables in the object table depends on the number of object versions that have been checked in with management object 42. Management objects 42 can perform blocks 103-105 since management objects 42 have information about stored interfaces which are distributed in each management object 42; Human manager object 44 reviews the results of blocks 103-105 from management object 42 and performs blocks 106 and 107.

Human management framework 19 can interact with life cycle framework 20. In specification stage 52, interface ids of objects 30 are specified by human manager object 44 and are forwarded to the developer by management object 42. The developer completes implementation stage 54 and then performs single-unit testing. Implemented objects are registered with management object 42. Interface ids, the associated interface version numbers, and the source and IDL files associated with object 30 wherein object 30 is registered are forwarded to management object 42. For example, Object 01 having two interface ids, 1 and 4, forwards the following information to management object 42, as shown in Table 18.

Table 18

ObjectID – 01 Information

InterfaceID	1	4
InterfaceVersion	1.0	1.0
IDL Description	01_Interface1.idl	01_Interface4.idl
SourceCodeFileName	01_Interface1.bas	01_Interface4.bas
LocalTypeName	StateObject1	BasicServicePortObject1

Management object 42 returns the appropriate object version number. Management object 42 keeps track of: for example, if the current version for object 30 being registered is already registered and this current registration is a newer implementation for the same combination of interface versions, in this case management object 42 returns the next immediate object version number, starting with the base object version for that combination of interface versions. If the current version for object 30 is not already registered, then for the new combination of interface versions, management object 42 returns a totally new object version for that combination, if human manager object 44 has not specified any increasing number for that interface version combination. Else, the management object 42 returns

a number assigned by human manager object 44. Failure of the registration process results in developer retrying registration.

At the beginning of the specification and object-oriented analysis stage 52, global interface ids for each object 30 are specified into the management object 42. The corresponding developers get this information in the beginning from management object 42. For each object 30, human manager object 44 can create a most recent version (MRV) ObjectID table in management object 42 with column names as the interface ids associated with an Object ID. The range of values associated with a column is the interface version numbers associated with that interface. The MRV ObjectID table is updated every time there is a successful registration. For every unique combination of interface versions, management objects 42 keep only the most recent implementation in this table. This table is used by human manager object 44 to perform block 106 to execute a join operation on all shared interfaces. For every table created by the human manager object 44 of objects 30, management objects 42 create another table referred to as ALLVERSIONS ObjectID table associated with the ObjectID and keep all the object versions associated with that object in it. This table is not known to human manager object 44. When using the increasing object version numbers policy, human manager object 44 specifies the increasing object version number for each unique combination of the interface versions to management objects. Management objects 42 update this information in the MRV ObjectID table. Accordingly, all the interface versions and the choice of human manager object 44 for the starting object version number are entered into the MRV ObjectID table.

Preferably, for every registration with new combination of interface versions, management objects 42 return the next number in the series 1, 2, 3, 4, 5, ... n, as object versions. The next number that will be issued will be found by management objects 42 from the MRV ObjectID table. For example, management object 42 returns 1.0 for the first unique combination of interface versions inputted. For subsequent multiple implementations of the same combination of interface versions, management object 42 returns 1.1, 1.2, 1.3, ... 1.n. When a second unique combination of interface versions is inputted, the management object 42 returns 2.0.

If there is no record for the MRV, it means that it is totally a new combination of interface versions, and human manager object 44 has not specified any starting object version number. For that combination, management objects 42 return the next number in the series 1, 2, ... by finding out which number has already been used thus far starting from 1.0 (from the MRV ObjectID table). Management objects 42 update the MRV ObjectID table and the ALLVERSIONS ObjectID table.

If there exists a most recent version and MRV ObjectVersion in the MRV Object ID table is equal to –1, it is totally a new combination of interface versions, but human manager object 44 has specified a starting object version number for that combination. (MRV ObjectVersion is left –1 to indicate no version exists so far). In this case, management object 42 returns a number specified by human manager object 44 to the developer. Management objects 42 modify the MRV ObjectID table tuple MRV_ObjectVersion (old is –1) to the human manager 44 specified number (indicating the first most recent version number). Thereafter, ALLVERSIONS ObjectID table is updated with a new tuple containing the object version information.

If there exists a most recent version and MRV ObjectVersion not equal to –1, it is a repeated combination of interface versions. For this combination, a look up is performed to find out the object version in the MRV ObjectID table. The lookup will return the most recent version number of the object version. For example, it could be 1.0, or 1.1, or 1.2, etc., such that adding 0.1 to that number gives the version number. The MRV ObjectVersion entry in the MRV ObjectID table is modified to reflect the newer version number obtained. The ALLVERSIONS ObjectID table is also updated with the newer version numbers. The version number is returned to the developer by management object 42.

A set of object versions are 'interface compatible' if all of them have the same interface version number associated with that interface. A set of object versions form a 'compatible application' if with respect to each shared interface, the connected set of object versions are interface compatible. In order to obtain all the unique compatible systems (partial order), all the MRV ObjectID tables should be joined, whenever all the columns with the same interface ids are equal and relevant object versions and interface versions should be projected (number of columns in the output is equal to number of objects plus the number of interfaces in the application). Accordingly, the inner join performed is an AND condition with respect to multiple shared interfaces. Human manager object 44 can make up a SQL query for the same and send it to management object 42. Management object 42 executes it and returns all the tuples to human manager object 44. From the partial order of all compatible systems, the human manager has the discretion of selecting a compatible system for deployment.

In an implementation of the distributed management network, human manager object 44 writes separate SQL strings for each management object 42 domain. Each SQL string includes only the local objects, but human manager object 44 obtains both the local as well as the shared interfaces. Each

management object 42 cannot make complete decisions about its outgoing interfaces. Each management object 42 executes the SQL string and return the results to human manager object 44. After obtaining response from all management objects 42, human manager object 44 performs a higher level join of all the intermediate results, with respect to all shared interfaces to obtain corresponding object and interface versions. The complexity of the distributed selection algorithm is *NLog(N)*, where N is the number of object versions (selectivity for a specific interface version value is bounded by a constant).

The following describe examples of the commands supported for human manager object 44.

1. CreateObject/DestroyObject: Human manager object 44 sends this command to management object 42 along with the ObjectID, ObjectVersion, and StartPortID. ObjectID and ObjectVersion uniquely identify the object 30 program to run. Management object 42 looks into its own database where it has stored information about that ObjectID and ObjectVersion. Management object 42 would have updated this database information at the time of developers' successful check-in. Management object 42 will extract the executable information, arguments, and copy all the DLL files (if any) to the appropriate directory. Using the OS-level system command, object 30 is executed along with the command line arguments. The first two arguments are: the hostname where the clock server is running, and the socket id where the parent clock is polling for children connections. The third argument is the StartPortID which represents object 30 port name space in the entire application network. No two objects 30 shall have the same port name space. Upon receipt of successful creation of object 30 from the low-level system, management object 42 returns a successful message to the manager 44. Otherwise, it returns a failure. Destroy object is achieved using a kill/die protocol through the state port (using an appropriate set operation).

2. CreatePort/DestroyPort: This command is used to create object ports 32 in the management object (by the management object) to tune them dynamically to different interactions of application. Object ports 32 are used to obtain states of objects 30 and the clocks. Ports 32 of management object 42 can be used to disable/enable the interactions transparently from outside. Disabling/Enabling the interactions is achieved using HoldClock/Release commands described below. Human manager object 44 sends PortID to be created, InterfaceID, and InterfaceVersion. The last two values are used to extract the size of object port 32, using an IDL description associated with that interface version, and an IDL Backend that generates size and offset of different elements in the structure. CreatePort method has two parameters: PortID representing the ServiceID and the size of object port 32. Size of

object port 32 depends upon the InterfaceID, InterfaceVersion and upon the architecture. Object ports 32 created are registered with management object 42 so that the human manager 44 can access its services by holding the PortID. On the same lines, object ports 32 can be destroyed when their services are not needed anymore by the network application.

3. CreateClock/DestroyClock: This command is used to create a clock in the management object 42. It has only one parameter: ClockID, which must be unique in environment 10. Once the clock with that ClockID is created, it is registered with the management system to access its services. When the clock is created, it comes with the base phase. Any object ports 32 tuned to that phase immediately get the first advance. After getting the first advance, one can create additional phases and tune more object ports 32 (which is application specific). A clock can be destroyed too, if its services are not needed anymore by the application.

4. AddPhase/Destroy Clock Phase: This command is used to create a refined phase dynamically with respect to the base phase of the clock. Then object ports 32 are tuned to that refined phase. The behavior of the application is changed by adding newer phases for observation and control. Similarly, clock phases can be destroyed. Add phase method has two parameters: Base ClockID and the RefinedClockID. Each phase of the clock is a separate service provided by clock and human manager object 44 access them separately depending on the requirements.

5. Tune/Detune Port: This command is used to tune (connect) different object ports 32 to different phases of the different clocks in the application. This is application specific and needs three parameters: PortID, ClockID, PhaseHandle (Base phase means ClockID, for any other phase use the RefinedClockID specified as above). Object ports 32 can be detuned when their roles are completed as part of different interactions.

6. Set/Get State of objects: For setting/getting states of objects 30, human manager 44 must first identify the state InterfaceIDs and their InterfaceVersion. After getting the IDL description for an object's state interface, human manager objects 44 give it to the IDL Backend. The backend IDL compiler generates FieldNumber, FieldType, FieldOffset, and FieldSize for each element of all the interface structures. It also generates the actual size of object ports 32. Human manager 47 needs to understand the individual elements of data bag 35 and the state. This feature is necessary to enable transparent access to the interactions. For example, for the InterfaceID = 1, 2, 3, the IDL backend outputs:

FieldNumber	FieldType	FieldName	FieldSize	FieldOffset	Data
1	INT	MessageType	2	0	1
2	INT	StateValue	2	2	0

For setting the state, human manager object 44 can write into the Data field of MessageType 1. Then human manager object 44 writes StateValue. The information about 1, 0 etc. is obtained from the state's IDL. Thus human manager object 44 makes up the above raw bytes (4 bytes of information in the above example). Then Set command is sent to management object 42 connected to that object's state interface, along with the PortID raw bytes data, and the size of raw bytes. The management object 32 could also obtain the size of raw bytes by receiving the object's state InterfaceID and InterfaceVersion. Once management object port 32 receives a set command, it writes that to object 30 through its object port 32 upon advance. When it receives an acknowledgment from object 30, management object 42 sends an acknowledgment back to human manager object 44.

For getting the state, human manager object 44 can write into the Data field of MessageType 2. The information about 2 is obtained from the state's IDL. The same steps as for setting the state are executed. In addition, human manager object 44 uses the previous technique to split the raw 4 bytes received from management object 42 into appropriate meaningful data. Methods have been supported which enable human manager 47 to see meaningful data. For example, human manager 47 could see the following:

FieldNumber	FieldType	FieldName	FieldSize	FieldOffset	Data
1	INT	MessageType	2	0	4
2	INT	StateValue	2	2	1

MessageType = 4 indicates Get ack and the state value 1 indicates the object state.

7. SetState/GetState of Clocks (links) (CollectBag, WriteBag): Instead of structures associated with state interfaces of objects 30, use the IDL structures associated with different interactions. For example, use the InterfaceID = 4, InterfaceVersion = 1.0. The IDL Backend generates the following:

FieldNumber	FieldType	FieldName	FieldSize	FieldOffset	Data
1	INT	MessageType	2	0	1
2	INT	Data	2	2	22

MessageType = 1 indicates OM_SQUARE operation. In case of interactions, the data bag may have more than one element and hence the read from and write to the clock (interfaces) may have multiple cycles.

8. Hold/Release Clock: HoldClock command tells management object 42 to hold the next advance at the specified object port 32, until notified by human manager object 44. Management object 42 will not release object port 32 unless informed by human manager object 44. Object ports 32 connected to other different phases in the same clock do not get advance and hence remain in quiescent stable states. Human manager object 44 can hold the advance signal as long as it needs, to analyze the application state. The hold is released by a specific release clock command from human manager object 44. Only when management object 42 sends the release along that port object 32, object ports 32 tuned to the next higher or lower phase are enabled to communicate with each other.

9. GetIDLs of Interfaces: Using this command human manager object 44 can get the IDL descriptions of objects 30 associated with different interfaces. When human manager object 44 receives IDL descriptions from management object 42, they are copies into a local database. This features enables human managers 47 to view the IDLs whenever they want.

10. Register/Deregister ports: These commands enable object ports 32 created in objects 30 to be registered with management object 42. Only then, human manager object 44 can access its services. Human manager object 44 defines the port name space for each of objects 30 in the application system. Port services can be deregistered when not needed by the application anymore.

Human management framework 19 can be used to consistently and transparently set up an application. Fig. 12 illustrates an example of a network application set up by human manager object 44. Human manager object 44 sets up network application 109, specifies the initial conditions, initiates and analyzes changes and controls evolution. Human manager object 44 decides about the state of Objects 01, 02 and 03 and associates state object ports 61a-c and basic service object ports 62a-c with respective Objects 01, 02 and 03. Human manager object 44 receives all IDL descriptions for interfaces of Objects 01, 02 and 03. The backend IDL compiler can provide a meaningful

interpretation of the received modified CORBA IDL. Fig. 13 is a flow diagram of a method for setting up a transaction network 110. Before starting setting up of the network application, the INE is made active, as shown in block 111. Referring to Fig. 12, INE interface 45 is activated. Human manager object 44 and management objects 42 can execute the following methods to create and register a respective INE port.

OMF_CreatePort(ManagementINEPortID, Sizeof(Struct INE)) //Port 1 in the management
OMF_Register(ManagementINEPortID)
OMF_CreatePort(ManagerINEPortID, SizeOf(StructINE))
OMF_Register(ManagerINEPortID)

Referring to Fig. 13 in block 112, a clock is created and registered with human manager object 44 and management objects 42. Clock 86m is created by human manager object 44 and registered with human manager object 44 and management object 42, as shown in Fig. 12. For example, a clock with ClockID = 50 is created with the command OMF_CreateClock(ClockID). INE ports 85a are tuned to the base phase of clock 86m using the following commands:

OMF_TunePort2Clock(ManagementINEPortID (Port 1), ClockID, ClockID);
OMF_TunePort2Clock(ManagerINEPortID, ClockID,ClockID)

In block 113, human manager object 44 instructs management object 42 to create ports for each interface. In network application 109, for example, four ports are created, each associated with respective InterfaceIDs 1, 2, 3 and 4. Human manager object 44 sends the InterfaceID and Interface Version. Management object ports are assigned Port 2, Port 3, Port 4 and Port 5. The size of management object ports 2-5 is established by human management object 44. For example, management object ports 2-5 can each have a size of 4 bytes which value is application specific.

In block 114, human manager object 44 instructs management objects 42 to create objects. For example, human manager object 44 sends a message to the management object 42 to create 3 objects: ObjectID is 1, ObjectVer is 1.0; ObjectID is 2, ObjectVer is 1.0; ObjectID is 3, ObjectVer 1.0. Each of the 3 objects create and register their corresponding state object ports 61a-c and basic service object ports 62a-c.

In block 115, management object 42 creates clocks for connecting a respective object state port 61a-c with a management object states Port 2-4. For example, ClockID = 1 is created to connect state port of object 01 and management state port Port 2 created for Object 01. ClockID = 2 is created to connect state port of Object 02 and management state Port 3 created for object 02. ClockID = 3 is

created to connect state port of Object 03 and management state port Port 4 created for object 03. ClockID = 4 is created to connect management state Port 5 for the InterfaceID = 4 to the base phase.

In block 116, human manager object 44 tunes object state ports and management state ports 2-4. For example, Object 01's state port and management state port Port 2 are tuned to the base phase of ClockID = 1. Object 02's state port and management state port Port 3 are tuned to the base phase of ClockID = 2. Object 03's state port and management state port Port 4 are tuned to the base phase of ClockID = 3. Management state port Port 5 is tuned to the base phase of ClockID = 4.

In block 117, human manager 44 initializes states of objects 30. For example, human manager 44 makes up a raw data of 4 bytes (using IDL output) to set states of objects 30 initially to zero and sends to management object 42 through the INE interface 45. Management object 42 writes the 4 bytes to the appropriate port that is connected to the corresponding management state port. The acknowledgment received by management object 42 is sent to human manager object 44, back through INE 45. The same is repeated for states of all objects.

Human manager object 44 specifies to hold the clock associated with management state Port 5 of management object 42. At the next immediate port advance, the clock is held by management object 42. It does not release the current step. It sends an acknowledgment to human manager object 44. Thereafter, human manager object 44 can add a phase and tune the three application service ports 62a-c to the new phase of the clock dynamically. Human manager object 44 instructs the management object 42 to release the clock. After management object 42 executes that command from human manager object 44, the application service ports get the advance and the null bag to begin with. At this time object 01 can send any request to any other object 02, 03. Objects which have been tuned to the base phase must forward the data and keep the port released always, unless there is a request by human manager object 44 to hold the clock, thereby ensuring continued progress of the application at this time. Application set up is complete. Human manager object 44 can continue to monitor the object states and the states of the clocks. Management object ports 42 can be detuned dynamically from the application, until there is a need to dynamic reconfiguration as described below.

Dynamic reconfiguration of a network application includes implementing new object versions adding and deleting object state and service ports in an application network without shutting down the services of existing objects. For example, bug fixing can result in newer versions. Object ports can be tuned or detuned for example if the independent transaction is complete or at an intermediate point. In the method of the present invention dynamic reconfiguration occurs at quiescent points of the

application. Quiescent points are established by management object 42 by following interaction framework 16 in which management object 42 receives advance only at all object ports base released and at least one port has triggered. The receipt of an advance by management object 42 is a quiescent point. Accordingly, quiescence is achieved automatically without forcing in a quiescent state allowing objects to send out messages voluntarily, collecting the messages by the clock and sending to the management object and waiting for management to get an advance from the clock before incorporating any changes.

Fig. 12B illustrates minimization of disruption to achieve quiescence. Application objects are not forced to go to quiescent states. Objects 01, 02, and 03 send out responses voluntarily. Clock collects the data and sends it to the management. Management waits for an advance from the clock. This reduces the complexity of the management. Human manager can incorporate changes only when the management gets the advance and has the control. Holding is done only upon receipt of advance. Managers transparently understand completion of independent transactions, from outside.

Fig. 14 illustrates a schematic diagram network application 109 in which a new version of an object is dynamically implemented. The existing object 02 has two active interfaces: the state interface 2 and the BasicServicePort interface 4. Object 02 must be quiescent with respect to both these interfaces, before detuning them from their respective clock phases. First, human manager object 44 sends the hold clock associated with InterfaceID = 4 message to management object 42. At the next immediate advance of management port Port 5 connected to the InterfaceID = 4, the clock is held by management object 42. Human manager object 44 gets the notification about the clock held. Upon notification, human manager object 44 gets data bag 35 (not shown). From the IDL backend compiler generated size, type, and offsets, human manager object 44 can understand the structure elements in data bag 35. The information obtained by human manager object 44 is the initial condition of the clock. At this point, since management object 42 is holding the advance of the clock associated with InterfaceID = 4, Objects 01, 02 and 03 already voluntarily sent releases, and at least one trigger for the previous step and are expecting the next advance to come at the earliest. At this point, object states 01, 02 and 03 are stable with respect to this interface. Objects 01, 02 and 03 do not know anything about to whom the messages are being passed in between steps, rather they voluntarily give out data, release, and trigger (if necessary) and management obtaining information is transparent to the affected objects. Human manager object 44 can decode the raw bytes received from

management object 42 with respect to the shared InterfaceID = 4 using the IDL Backend. For example, the output could be one of the following:

Situation #1. MessageType = OM_SQUARE; Data = UserInputInteger;

Situation #2. MessageType = OM_SUMMATION; Data = UserInputInteger;

Situation #3. Messagetype = OM_SQUARE_OUTPUT; Data = Squared UserInputInteger;

Situation #4. MessageType = OM_SQUARE_SUMMATION:

Data = Summation(UserInputInteger);

The goal is to replace Object 02 version 1.0 by Object 02 Version 1.1. Depending upon the situation, human manager object 44 must map the situation to the IDL description associated with the current running version and understand how the transaction is proceeding and where it is at this point of time. Human manager object 44 must find out if the inputs affect the states of objects. From the IDL of Object 02 the following scenerios can be established:

1. If it is situation #1, then Object 01 has just initiated the transaction; Object 01's state is 1; Object 02's state is 0; Object 03's state is 0; if the older and newer object versions have identical state transitions, the older version can be replaced with the newer one and the newer version of Object 02 can be initialized to 0. After that, let the transaction continue and human manager object 44 must release the held clock. If the older and newer object versions do not have identical state transitions, then human manager object 44 will have to find an equivalent state in the newer version and initialize it. If human manager object 44 cannot find an equivalent state, then let the transaction proceed for one more step and then hold it again. Repeat the previous step, until the independent transaction completes at the initiator. At the end of the transaction, all the objects can be replaced if needed.

2. If it is situation #2 or #3 or #4, Object 02's state is 0; if the older and newer object versions have identical state transitions, the older version can be replaced now with the newer one, and the newer version of Object 02 must be initialized to 0. After that, let the transaction continue and the human manager object must release the held clock. If the older and newer object versions do not have identical state transitions, then human manager object 44 will have to find an equivalent state in the newer version and initialize it. If human manager object 44 cannot find an equivalent state, then let the transaction proceed for one more step and then hold it again. Repeat the previous step, until the independent transaction completes at the initiator. At the end of the transaction, all the objects can be replaced if needed.

The conclusions that human manager 47 can draw by mapping the situation to the IDL description is that older Object 02 can be replaced at any step in the transaction: TRANSAC_NAME_SQUARESUM. If human manager object 44 cannot find a mapping state in the newer object version, then let the transaction go by another cycle and then hold it again. Repeat checking and this process until the transaction ends at the initiator. Also find equivalent states when needed. In this example, human manager object 44 was able to conclude states of object 01, 02 and 03 by looking at the input. Alternatively, human manager object 44 will need the states of the objects and the links before deciding status of objects 30.

For example, if it is scenario #1, human manager object 44 finds that newer object version has a state which could be initialized to 0 and it's behavior is identical to the older version. Human manager object 44 notifies management object 42 to detune object 02 service port from the clock. Management object 42 executes it. Object 02 cannot exchange any more advance or data. Object 01 and Object 03 also cannot receive advance or data and hence cannot communicate anymore. Human manager object 44 sends a message to management object 42 to get the state of Object 02. Management object 42 sends the get state request to the object state interface of Object 02. Object 02 divulges the stable state and management object 42 receives at Port 3.

Management object 42 returns that state information to human manager object 44 and again human manager object 44 understands the actual state information. Human manager object 44 sends a message to destroy object 02. The destroy message is sent to object 02 and an ack is received by management object 42. At this time, the state port of old object 02 and clockID = 2 are destroyed that connects the state port of 02 and the management object Port 3. The old object 02 destroys itself and also Basic service port 62b upon receipt of this message. Human manager object 44 is notified about the destroy. It is noted that human manager object 44 is still holding the ClockID = 4 at the base phase. Management object Port 3 is destroyed.

Human manager object 44 creates the new version of the object, for example 1.1. Human manager object 44 creates a port in the management object Port 6 and creates a clock with ClockID = 5, and then tunes the new object's state port, and the new management port Port 6 created to the base phase of the clock. Human manager object 44 sets the new object state to the value obtained from the old object if there is a direct one-to-one state mapping, otherwise it can set to an equivalent state in the new object. Afterwards, the service port of the new object is tuned to the phase 1 of the clock. The clock data that the manager had received immediately after holding the clock, is written back to the

clock by human manager object 44 and human manager object 44 releases the clock. Then all the three object ports for objects 01, 02 and 03 get an advance. Object 02 will immediately receive the request from object 01 that was in transit which was held by human manager object 44. The application continues as though it was the initial configuration. Neighboring object 01 and object 03 do not know anything about object 02's replacement.

Fig. 15 illustrates a schematic diagram of dynamically implementing a new version of an interface. For example, in InterfaceID = 4 can be changed to include third field to denote objectID indicating from which object it is coming.

First, human manager object 44 sends the hold clock associated with InterfaceID = 4 message to the management object 42. At the next immediate advance of the management object Port 5 connected to the InterfaceID = 4, the clock is held by management object 42. Human manager object 44 gets the notification about the clock held. Upon notification, human manager object 44 gets the data in data bag 35 (not shown) and understands it. This information obtained by the human manager object 44 is the initial condition of the clock. At this point, since management object 42 is holding the advance of the clock associated with InterfaceID = 4, Objects 01, 02 and 03 already sent voluntarily all releases, and at least one trigger for the previous step and are expecting the next advance to come at the earliest.

As explained above, there could be four situations. Since there is a need to replace all the three, the transaction must be completed at the initiator object 01, or equivalent states must be found. In this case, all the objects are destroyed and newer object versions with the newer interface version are put into the system. Depending upon the situation, human manager object 44 notifies management object 42 to detune object 02 service port from the clock. Management object 42 executes it. Object 02 cannot exchange any more advance or data. Object 01 and object 03 too cannot receive advance or data and hence cannot communicate anymore. Human manager object 44 sends a message to the management object 42 to get the state of object 02. Management object 42 sends the get state request to the object state interface of object 02. Object 02 divulges the stable state and the management object 42 receives at Port 3.

Similarly service ports of old object 01 and object 03 are detuned from the clock. Human manager object 42 gets the states of object 01 and object 03. Management object 42 returns that state information to human manager object 44 and again the human manager object 44 understands the actual state information.

Human manager object 44 sends a message to destroy object 02. The destroy message is sent to the object and an ack is received by the management object 42. The state port of old Object 02 is detuned then. ClockID = 2 is destroyed that connects the state port of object 02 and the management object Port 3 created for object 02. The old object 02 destroys itself and also the basic service port upon receipt of this message. Human manager object 44 is notified about the destroy. The same steps are repeated for objects 01 and 03. It is noted that management object 42 is still holding the ClockID = 4 at the base phase. Management port Ports 2, 3, and 4 are destroyed.

Human manager object 44 creates the new version of the object 01's version 1.1. Human manager object 44 creates a port in the management object Port 6 and creates a clock with ClockID = 5, and then tunes the new object's state port, and the new management port created to the base phase of the clock. Human manager object 44 sets the new object state to the value obtained from the old object if there is a direct one-to-one states, otherwise may set to an equivalent state in the new object. The same is repeated for object 02 and object 03. The states are set accordingly by human manager object 44. Now human manager object 44 creates Port 9 of size 6 bytes. Human manager object 44 creates the clock and tunes the Port 9 to the base of that clock. Human manager object 44 specifies to management object 42 to hold that clock. At the arrival of the next advance, management object 42 notifies human manager object 44 about it. Human manager object 44 can create phase 1 in the clock, tune all the three new Basic service ports of newer objects to that phase, and then release the clock.

The clock data that human manager object 44 had received immediately after holding the clock 4 had only 4 bytes of information. But the new clock has 6 bytes information. Human manager object 44 has to translate the initial conditions of the interfaces accordingly so that evolving application is consistent. Human manager object 44 can modify the old clock data and then it is written back to the clock 8 through management object 42. Afterwards, human manager object 44 releases the clock. Only now, that all the 3 ports of the objects get an advance and human manager has written data bag 35, the application continues as though it was the initial configuration. None of the old objects 01, 02 and 03 participate.

Dependent transactions are handled in the same way independent transactions are handled by human managers object 44. Human manager object 44 must wait until the management port tuned to an interface (which supports a dependent transaction) receives an advance. The management object port will receive an advance only after all the other dependent and/or independent transactions associated with this interface complete. The advance may be received after quite sometime, but

advance will be received (since both the dependent and the independent transactions must complete in a finite time at the initiator). Finally, when the above management port advances, the human manager can hold it, analyze the conditions and then do the appropriate changes. A dependency graph can be used for determining states of objects. The human managers can analyze the states of all the objects by the dependency graph. In the IDL descriptions, instead of using independent transactions, the descriptions show the dependencies.

It is to be understood that the above-described embodiments are illustrative of only a few of the many possible specific embodiments which can represent applications of the principles of the invention. Numerous and varied other arrangements can be readily devised in accordance with these principles by those skilled in the art without departing from the spirit and scope of the invention.

What is claimed is:

1. A distributed object-oriented software development environment comprising:

a plurality of objects for performing object operations, each object including an object interface;

at least one object port coupled to said each object interface of said objects; and

interaction means for connecting said object port of one of said objects to said object port of another one of said objects,

wherein one of said objects can communicate to another one of said objects if said object interfaces are compatible and said interaction means provides sequential flow of data and control from said object operations through a dynamically varying set of said ports having said compatible interfaces.

2. The environment of claim 1 wherein said interaction means is represented by a circular communication pathway and a first said object port is connected to said circular communication pathway to receive communications from at least a second said object port which is connected to said circular communication pathway.

3. The environment of claim 1 wherein said interface is described in modified CORBA interface description language.

4. The environment of claim 1 further comprising:

a plurality of management objects, each said management object being associated with at least one of said objects;

a human manager object; and

an interface for network evolution for coupling said management objects to said human manager object, wherein said human manager object manages said objects through said management objects.

5. The environment of claim 4 wherein said human manager object assigns increasing object version numbers to said objects.

6. The environment of claim 5 wherein said human manager object assigns monotomically increasing interface versions to said object interfaces wherein each said object interface has a unique global identification in said application network.

7. The environment of claim 6 further comprising:

means for determining said compatible interfaces of said objects by registering said global identification and said object version number of said object with said management object.

8. The environment of claim 7 further comprising:

means for determining an object table comprising rows representing said object versions of said objects in said network application and columns representing an object identification and interface identification;

means for sorting said determined object table with respect to an object version;

means for sorting a first said sorted object table for a first said object and a second said sorted object table for a second said object with respect to a common said interface identification;

means for joining said first said sorted object table and said second said sorted object with respect to said interface identification; and

means for extracting said compatible object from said join of said object tables.

9. The environment of claim 8 further comprising:

means for sorting a subsequent object table with respect to said common said interface identification; and

means for joining said subsequent object table with said joined first said sorted object table and said second said sorted object table.

10. The environment of claim 1 further comprising a life cycle framework including a specification stage in which said objects and said interfaces are specified, a design stage in which said interfaces of said objects are negotiated, an implementation stage in which said

negotiated interfaces of said objects are implemented and a testing stage in which said implemented interfaces are tested.

11. A method for implementing negotiation during software development comprising the steps of:

a. determining a human manager object;

b. determining at least one management object;

c. determining an interface for network evolution (INE) between said human manager object and said management object, by said human manager object, by instructing objects with said at least one management object to create a plurality of objects for performing object operations, each said object including an object interface,

d. creating an interaction means for connecting said at least one object to said management objects;

e. determining at least one management object port associated with said management object;

f. determining at least one object port associated with said object; and

g. forwarding negotiations from said object ports to said management object ports.

12. The method of claim 11 further comprising the step of:

assigning tasks of designing said objects from said human manager object to a respective developers associated with at least one of said objects.

13. The method of claim 12 further comprising the step of:

creating a developer negotiation port by said developer for each of said objects to be developed.

14. The method of 13 further comprising the step of:

registering said developer negotiation ports with said human manager object.

15. The method of claim 14 further comprising:

creating management negotiation ports at said management objects which are each associated respectively with one of said developer negotiation ports.

16. The method of claim 15 wherein step g comprises:

forwarding negotiation scripts written in modified CORBA IDL by said developers through said respective developer negotiation ports to said respective manager negotiation ports for forwarding to designated objects.

17. The method of claim 16 further comprising the step of:

forwarding said scripts written in modified CORBA IDL received at said management object to said human manager object via said INE.

18. The method of claim 17 further comprising the step of:

interpreting said script written in modified CORBA IDL received at said human manager object into human readable data.

19. The method of claim 11 wherein the step of forwarding negotiations is repeated until all developers have agreed.

20. The method of claim 19 wherein said negotiations determine an object interface defined in modified CORBA IDL.

21. A method for implementing a network application comprising the steps of:

determining a plurality of objects;

associating an object port with each of said objects;

determining transactions for exchanging messages between said objects;

determining an object interface for each said object; and

implementing each determined object interface, wherein said messages are exchanged sequentially between said objects having compatible said object interfaces.

22. The method of claim 21 further comprising the step of:

registering said implemented object and said object interface with a management framework, said management framework returning an object identification and an object version identification and an interface version identification.

23. The method of claim 22 wherein said implementing step further comprises the step of:

determining a network application having compatible said object version identifications.

24. The method of claim 23 wherein said step of determining a network application having compatible object versions comprises the steps of:

a. determining an object table comprising rows representing said object identification and said object version identification and columns representing said interface version identification;

b. sorting said determined object table with respect to said object version identification;

c. sorting a first said sorted object table for a first said object and a second said sorted object table for a second said object with respect to a common said interface identification;

d. joining said first said sorted object table and said second said sorted object with respect to said interface identification to form a join of said object tables; and

e. extracting said compatible object from said join of said object tables.

25. The method of claim 24 further comprising the steps of:

f. sorting a subsequent object table with respect to said common said interface identification; and

g. joining said subsequent object table with said joined object table of step (d);

26. The method of claim 24 wherein said object tables are created to have said object version identification and said interface version identification increasing in said rows and said columns.

27. The method of claim 21 further comprising the steps of:

determining a plurality of management objects, each said management object being associated with at least one of said objects;

determining a human manager object;

determining an interface for network evolution for coupling said management objects to said human manager object; and

managing said objects by said human manager object through interacting with said management objects.

28. The method of claim 27 further comprising the steps of:

updating said determined objects; and

assigning increasing object version numbers by said human manager object to said updated objects through said management objects.

29. The method of claim 27 further comprising the step of:

updating said object interface; and

assigning increasing interface version numbers by said human manager object to said updated object through said management objects.

30. A method for setting up a network application comprising the steps of:

a. determining a human manager object;

b. determining at least one management object;

c. determining an interface for network evolution (INE) between said human manager object and said management object, by said human manager object; instructing said at least one management object by said human manager object to create at least one object for performing object operations, each said object including an object interface,

d. creating an interaction means for connecting said objects to said management objects, said interaction means also being connected to said INE and said human manager object; and

e. initializing states at said human manager object of said objects and forwarding said initialized states to said objects via said INE to forward to said initialized states to said

management object and said management object forwarding said initialized states from said management object to said objects.

31. The method of claim 30 after step c further comprising the steps of:

f. determining a human manager object INE port for said human manager object;

g. determining a management object INE port for said management object; and

h. associating said INE with said INE port for said management object and said INE port for said manager object.

32. The method of claim 31 further comprising the steps of:

determining at least one port associated with said management object; and

determining at least one object port associated with each said object.

33. The method of claim 30 wherein said object interface is defined in modified CORBA IDL.

34. A method for dynamically reconfiguring a network application comprising the steps of:

determining a human manager object;

determining at least one management object;

determining an interface for network evolution (INE) between said human manager object and said management object, by said human manager object, by instructing objects with said at least one management object to create at least one object for performing object operations, each said object including an object interface and having an original state,

creating an interaction means for connecting said at least one object to said management objects;

determining at least one management object port associated with said management object;

determining at least one object port associated with said object; and

establishing quiesent points in at least one of said objects to be reconfigured through said management object.

35. The method of claim 34 further comprising the step of:

forwarding data for updating said at least one object from said object to said human manager object.

36. The method of claim 35 further comprising the steps of:

determining said port of said object to be reconfigured;

sending a destroy command from said human manager object to destroy said port to be reconfigured;

creating a new version of said object to be reconfigured at said human manager object;

forwarding said new version of said object to said management object;

creating a new object having said new version of said object; and

determining a new object port associated with said new object.

37. The method of claim 36 further comprising the steps of:

determining at said human manager object if a said original state of said object is the same as a state of said new version of said object; and

if said original object version and said new version have the same states, replacing said original object version with said new version; or

if said original object version and said new version do not have the same state, determining at said human manager object an equivalent state and replacing said original version with said new version.

38. The method of claim 37 further comprising the step of:

forwarding data for updating said at least one interface version from one of said objects to said human manager object.

39. The method of claim 38 further comprising the steps of:

determining a number of said objects to be reconfigured for said updating of said interface version;

sending a destroy command from said human manager to destroy said number of objects to be reconfigured;

creating a new version of each said number of objects to be reconfigured at said human manager object;

forwarding said new versions to said management object; and

creating a corresponding number of new objects having said new versions.

40. The method of claim 34 wherein said object interface is defined in modified CORBA IDL.

41. A system for implementing negotiation during software development comprising:

means for determining a human manager object;

means for determining at least one management object;

means for determining an interface for network evolution (INE) between said human manager object and said management object, by said human manager object, by instructing objects with said at least one management object to create a plurality of objects for performing object operations, each said object including an object interface;

means for creating an interaction means for connecting said at least one object to said management objects;

means for determining at least one management object port associated with said management object;

means for determining at least one object port associated with said object; and

means for forwarding negotiations from said object ports to said management object ports.

42. The system of claim 41 further comprising:

means for creating a developer negotiation port by said developer for each of said objects to be developed.

43. The system of claim 42 further comprising:

means for registering said developer negotiation ports with said human manager object.

44. The system of claim 43 further comprising:

means for creating management negotiation ports at said management objects which are each associated respectively with one of said developer negotiation ports.

45. The system of claim 44 wherein said negotiations are written in modified CORBA IDL.

46. A system for implementing a network application comprising:

means for determining a plurality of objects;

means for associating an object port with each of said objects;

means for determining transactions for exchanging messages between said objects;

means for determining an object interface for each said object; and

means for implementing each determined object interface, wherein said messages are exchanged sequentially between said objects having compatible said object interfaces.

47. The system of claim 46 further comprising:

means for registering said implemented object and said object interface with a management framework, said management framework returning an object identification and an object version identification and an interface version identification.

48. The system of claim 47 wherein said means for implementing comprises:

means for determining a network application having compatible said object version identifications.

49. The system of claim 48 wherein said means for determining a network application having compatible object versions comprises:

means for determining an object table comprising rows representing said object identification and said object version identification and columns representing said interface version identification;

means for sorting said determined object table with respect to said object version identification;

means for sorting a first said sorted object table for a first said object and a second said sorted object table for a second said object with respect to a common said interface identification;

means for joining said first said sorted object table and said second said sorted object with respect to said interface identification to form a join of said object tables; and

means for extracting said compatible object from said join of said object tables.

50. The system of claim 46 wherein said object interface is defined in modified CORBA IDL.

51. A system for setting up a network application comprising:

means for determining a human manager object;

means for determining at least one management object;

means for determining an interface for network evolution (INE) between said human manager object and said management object, by said human manager object; instructing said at least one management object by said human manager object to create at least one object for performing object operations, each said object including an object interface,

means for creating an interaction means for connecting said objects to said management objects, said interaction means also being connected to said INE and said human manager object; and

means for initializing states at said human manager object of said objects and forwarding said initialized states to said objects via said INE to forward to said initialized states to said management object and said management object forwarding said initialized states from said management object to said objects.

52. The system of claim 51 further comprising:

means for determining a human manager object INE port for said human manager object;

means for determining a management object INE port for said management object; and

means for associating said INE with said INE port for said management object and said INE port for said manager object.

53. The system of claim 52 further comprising:

means for determining at least one port associated with said management object; and

means for determining at least one object port associated with each said object.

54. The system of claim 51 wherein said object interface is defined in modified CORBA IDL.

55. A system for dynamically reconfiguring a network application comprising:

means for determining a human manager object;

means for determining at least one management object;

means for determining an interface for network evolution (INE) between said human manager object and said management object, by said human manager object, by instructing objects with said at least one management object to create at least one object for performing object operations, each said object including an object interface and having an original state,

means for creating an interaction means for connecting said at least one object to said management objects;

means for determining at least one management object port associated with said management object;

means for determining at least one object port associated with said object; and

means for establishing quiesent points in at least one of said objects to be reconfigured through said management object.

56. The system of claim 55 further comprising:

means for forwarding data for updating said at least one object from said object to said human manager object.

means for determining said port of said object to be reconfigured;

means for sending a destroy command from said human manager object to destroy said port to be reconfigured;

means for creating a new version of said object to be reconfigured at said human manager object;

means for forwarding said new version of said object to said management object;

means for creating a new object having said new version of said object; and

means for determining a new object port associated with said new object.

57. The system of claim 56 further comprising:

means for forwarding data for updating said at least one interface version from one of said objects to said human manager;

means for determining a number of said objects to be reconfigured for said updating of said interface version;

means for sending a destroy command from said human manager to destroy said number of objects to be reconfigured;

means for creating a new version of each said number of objects to be reconfigured at said human manager object;

means for forwarding said new versions to said management object;

means for creating a corresponding number of new objects having said new versions.

58. The system of claim 55 wherein said object interface is defined in modified CORBA IDL.

PCT

WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 7 : G06F 9/45

A1

(11) International Publication Number: WO 00/67121

(43) International Publication Date: 9 November 2000 (09.11.00)

(21) International Application Number: PCT/US00/11428

(22) International Filing Date: 28 April 2000 (28.04.00)

(30) Priority Data:
60/131,506 29 April 1999 (29.04.99) US

(71) Applicant *(for all designated States except US)*: RUTGERS UNIVERSITY [US/US]; Office of Corporate Liaison & Technology Transfer, 58 Bevier Road, Piscataway, NJ 08854 (US).

(71)(72) Applicant and Inventor: SURYANARAYANA, Manjunath, M. [–/US]; 110A Cedar Lane, Highland Park, NJ (US).

(74) Agent: DUNN MCKAY, Diane; Mathews, Collins, Shepherd & Gould, P.A., Suite 306, 100 Thanet Circle, Princeton, NJ 08540 (US).

(81) Designated States: AE, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CR, CU, CZ, DE, DK, DM, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA, UG, US, UZ, VN, YU, ZA, ZW, ARIPO patent (GH, GM, KE, LS, MW, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

Published

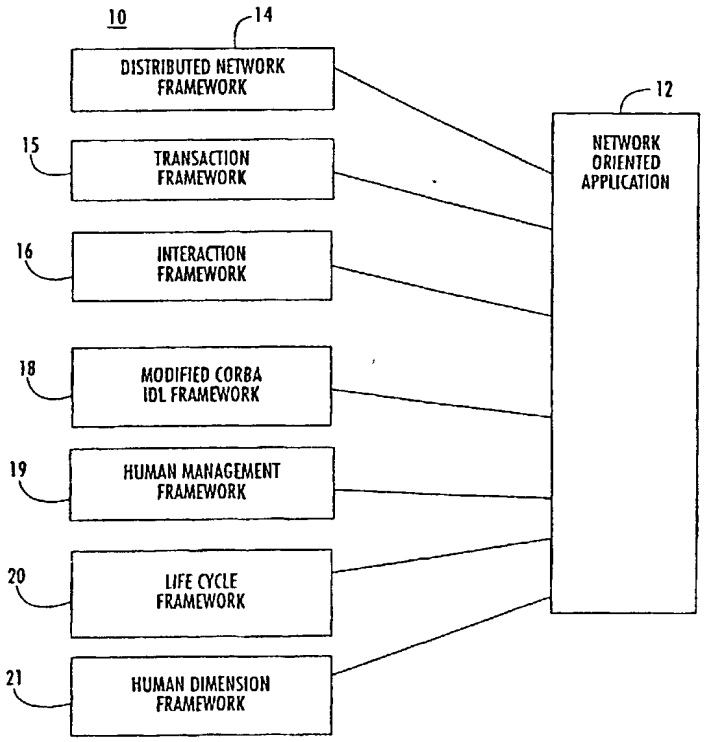
With international search report.

Before the expiration of the time limit for amending the claims and to be republished in the event of the receipt of amendments.

(54) Title: DISTRIBUTED SOFTWARE DEVELOPMENT ENVIRONMENT

(57) Abstract

The present invention relates to a distributed object–oriented software development environment (10). Objects for performing object operations can communicate to one another if associated interfaces are compatible. The environment provides sequential flow of control and data to the objects. Each interface of an object has a unique global identification value. Compatible interfaces are determined by registering the global identification and version of the interfaces with a management framework. The environment can be used for establishing an application network (12) and performing consistent and transparent dynamic updates of the application network. The management framework can include management objects associated with objects of the application. The management objects (42) communicate information about the objects to a human manager object (44). Accordingly, the human management object can control transactions of the objects during establishment or reconfiguration of the application network without participation by the objects. During a design stage, the management framework can also be used by the software developers for performing tetherless negotiation.



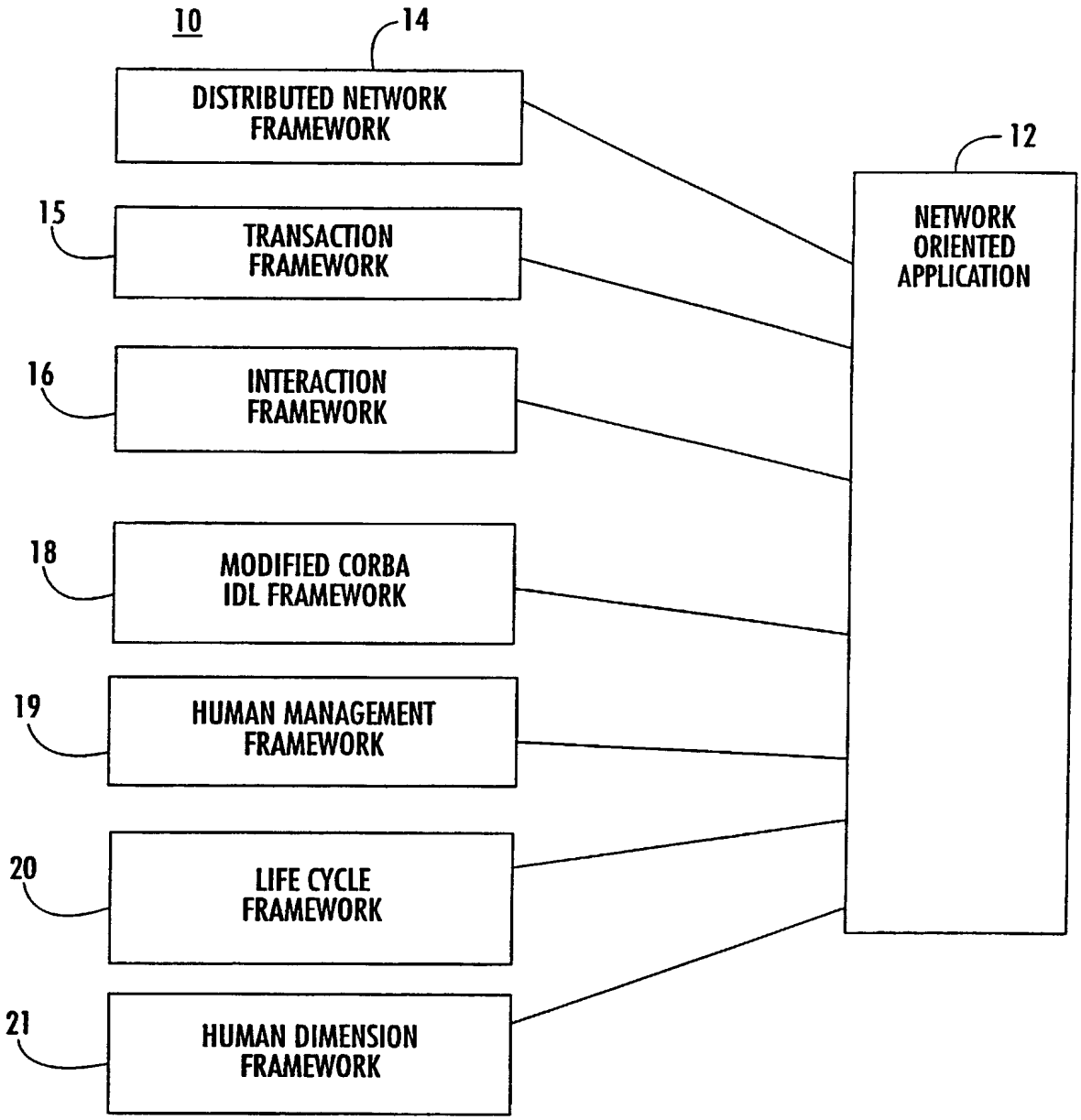

10
14
DISTRIBUTED NETWORK FRAMEWORK
15
TRANSACTION FRAMEWORK
16
INTERACTION FRAMEWORK
18
MODIFIED CORBA IDL FRAMEWORK
19
HUMAN MANAGEMENT FRAMEWORK
20
LIFE CYCLE FRAMEWORK
21
HUMAN DIMENSION FRAMEWORK
12
NETWORK ORIENTED APPLICATION

FIG. 1.

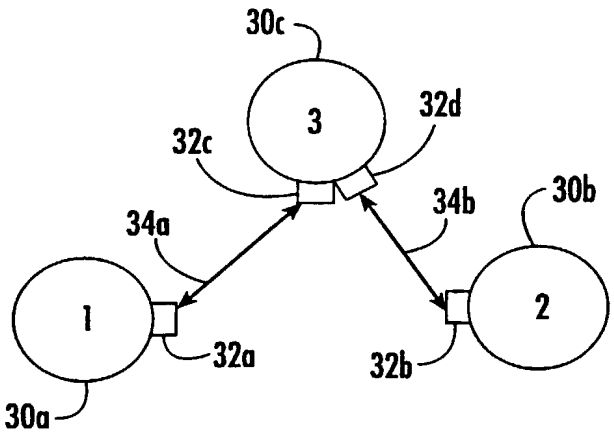


FIG. 2A.

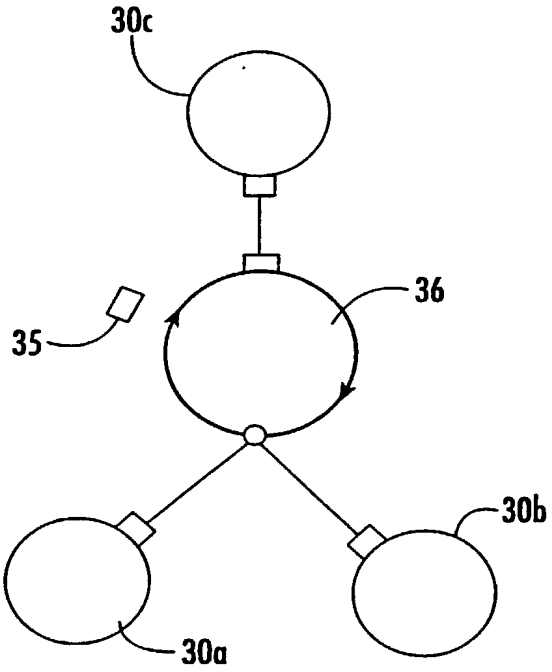


FIG. 2B.

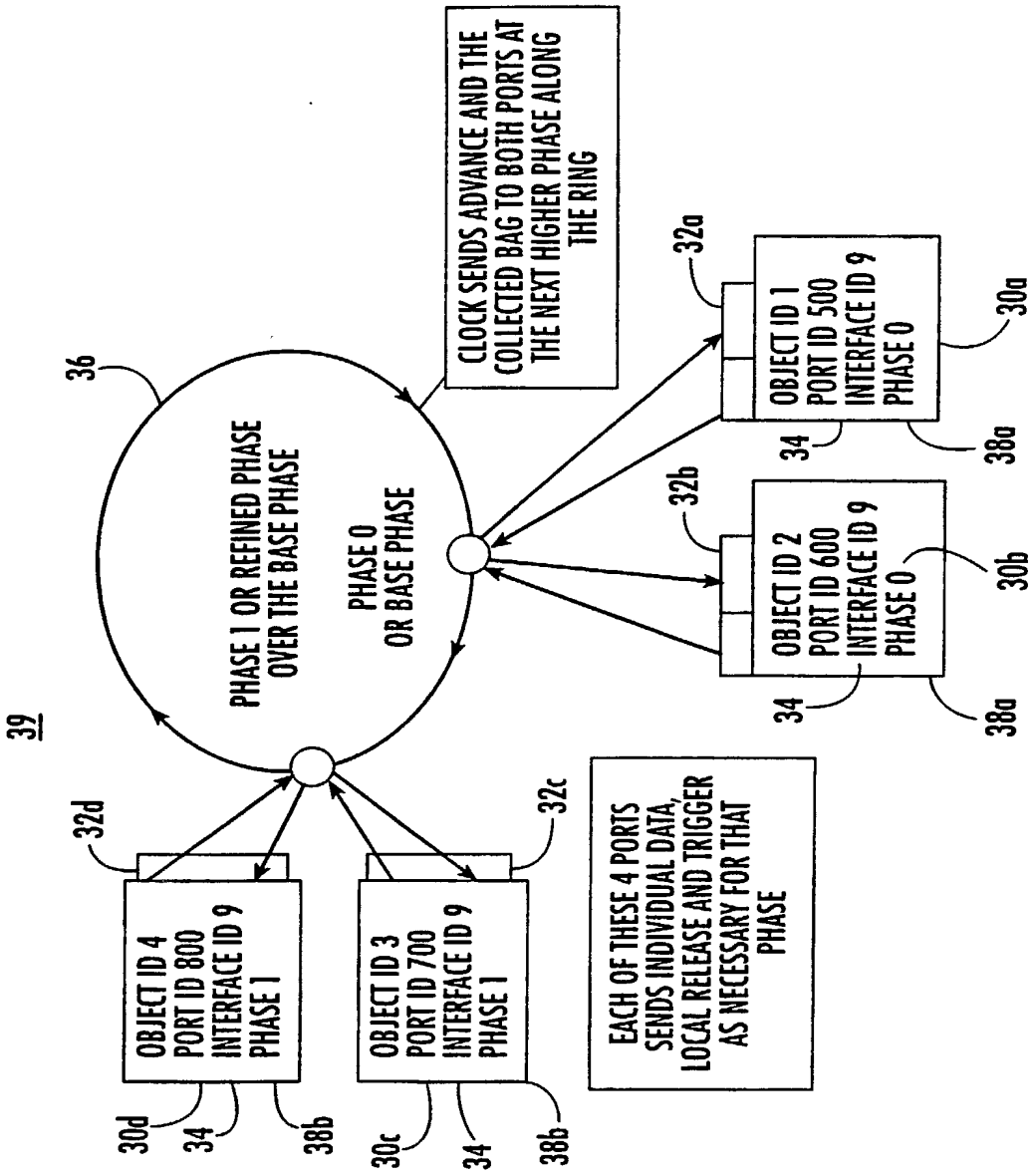

39
36
PHASE 1 OR REFINED PHASE OVER THE BASE PHASE
PHASE 0 OR BASE PHASE
CLOCK SENDS ADVANCE AND THE COLLECTED BAG TO BOTH PORTS AT THE NEXT HIGHER PHASE ALONG THE RING
32a
OBJECT ID 1 PORT ID 500 INTERFACE ID 9 PHASE 0
30a
34
38a
32b
OBJECT ID 2 PORT ID 600 INTERFACE ID 9 PHASE 0
30b
34
38a
32d
OBJECT ID 4 PORT ID 800 INTERFACE ID 9 PHASE 1
30d
34
38b
32c
OBJECT ID 3 PORT ID 700 INTERFACE ID 9 PHASE 1
30c
34
38b
EACH OF THESE 4 PORTS SENDS INDIVIDUAL DATA, LOCAL RELEASE AND TRIGGER AS NECESSARY FOR THAT PHASE

FIG. 3A.

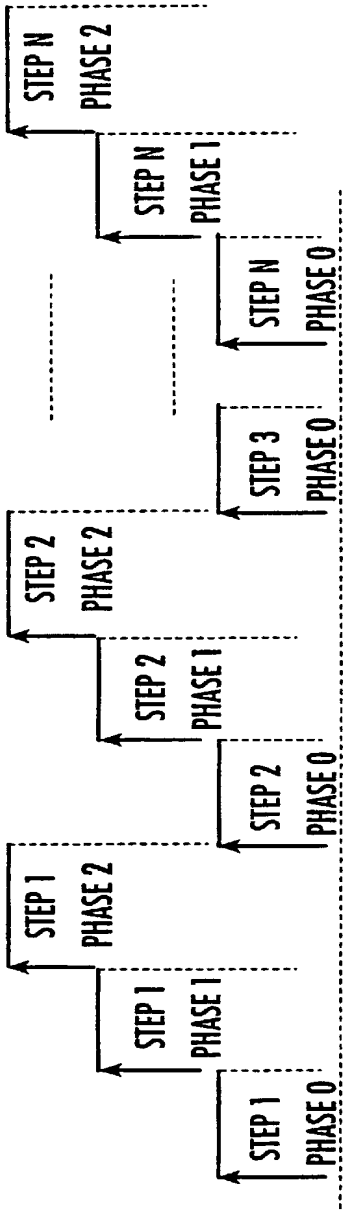

STEP N PHASE 2
STEP N PHASE 1
STEP N PHASE 0
STEP 3 PHASE 0
STEP 2 PHASE 2
STEP 2 PHASE 1
STEP 2 PHASE 0
STEP 1 PHASE 2
STEP 1 PHASE 1
STEP 1 PHASE 0
INDICATES ADVANCE TO THAT PHASE

FIG. 3B.

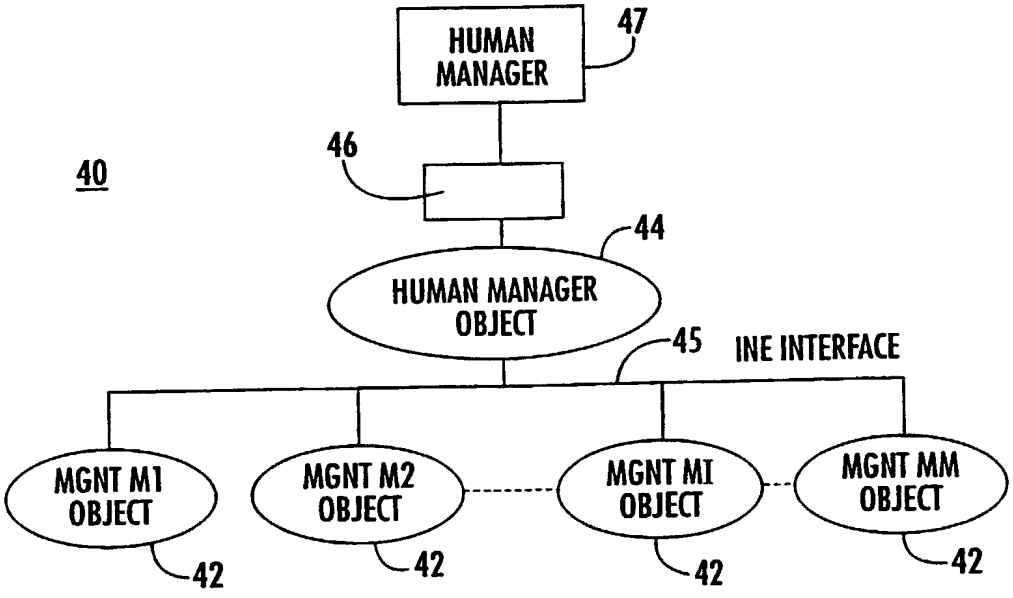
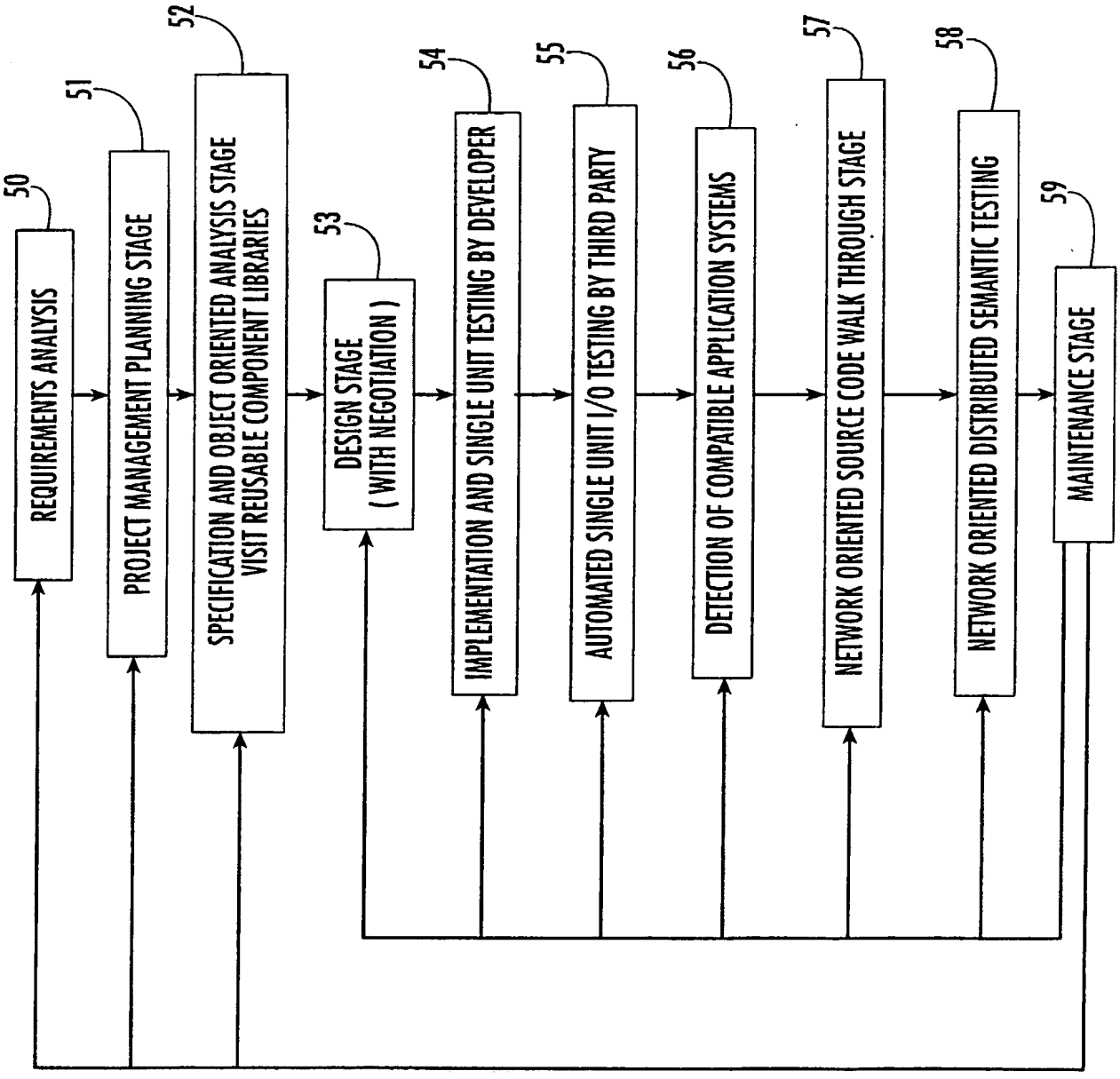

HUMAN MANAGER
47
46
40
44
HUMAN MANAGER OBJECT
45
INE INTERFACE
MGNT M1 OBJECT
MGNT M2 OBJECT
MGNT MI OBJECT
MGNT MM OBJECT
42
42
42
42

FIG. 4.

FIG. 5.


50
REQUIREMENTS ANALYSIS
51
PROJECT MANAGEMENT PLANNING STAGE
52
SPECIFICATION AND OBJECT ORIENTED ANALYSIS STAGE
VISIT REUSABLE COMPONENT LIBRARIES
53
DESIGN STAGE
(WITH NEGOTIATION)
54
IMPLEMENTATION AND SINGLE UNIT TESTING BY DEVELOPER
55
AUTOMATED SINGLE UNIT I/O TESTING BY THIRD PARTY
56
DETECTION OF COMPATIBLE APPLICATION SYSTEMS
57
NETWORK ORIENTED SOURCE CODE WALK THROUGH STAGE
58
NETWORK ORIENTED DISTRIBUTED SEMANTIC TESTING
59
MAINTENANCE STAGE

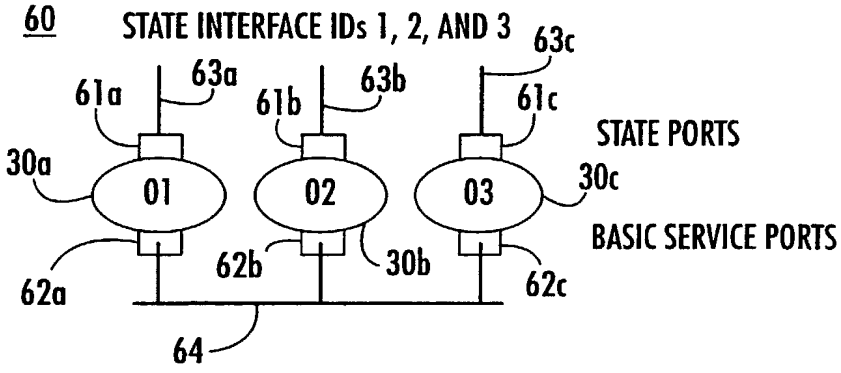

60
STATE INTERFACE IDs 1, 2, AND 3
63c
63a
63b
61a
61b
61c
STATE PORTS
30a
01
02
03
30c
BASIC SERVICE PORTS
62b
30b
62a
62c
64
GLOBAL COMMON INTERFACE ID = 4 CONNECTING ALL BASIC SERVICE OBJECT PORTS

FIG. 6.

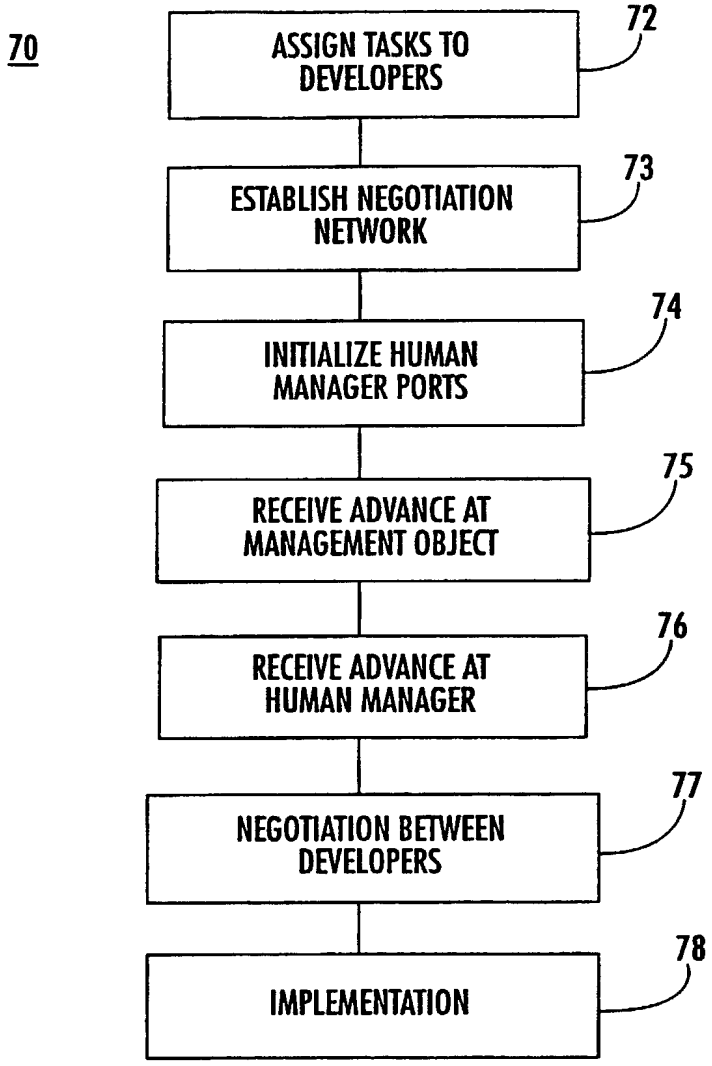

70
ASSIGN TASKS TO DEVELOPERS
72
ESTABLISH NEGOTIATION NETWORK
73
INITIALIZE HUMAN MANAGER PORTS
74
RECEIVE ADVANCE AT MANAGEMENT OBJECT
75
RECEIVE ADVANCE AT HUMAN MANAGER
76
NEGOTIATION BETWEEN DEVELOPERS
77
IMPLEMENTATION
78

FIG. 7.

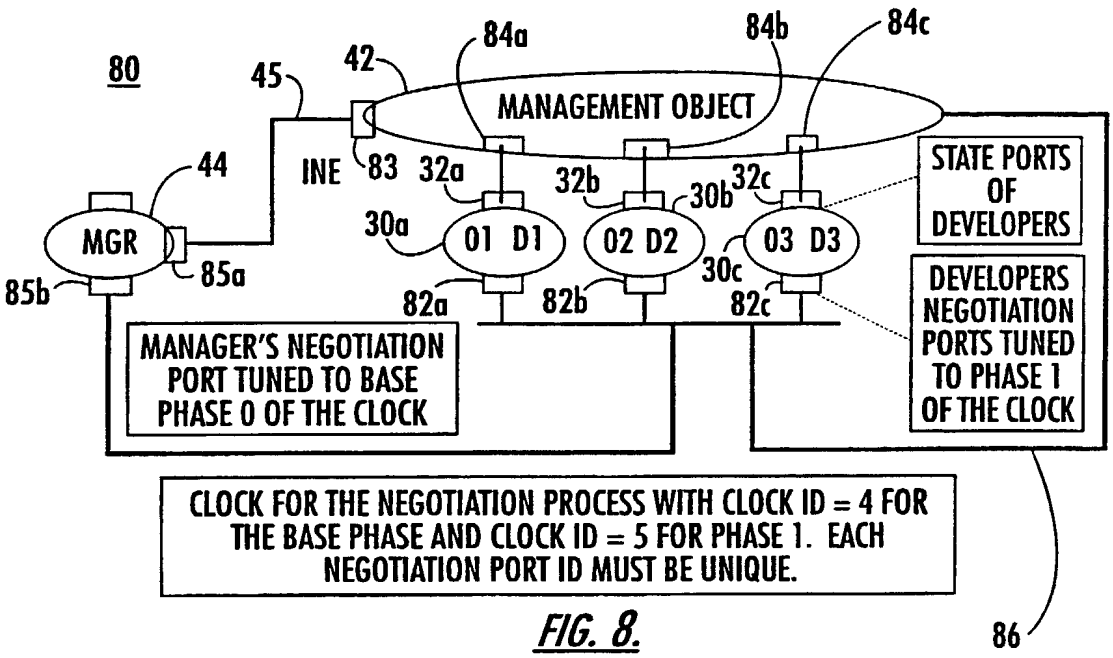

80
45
42
84a
84b
84c
MANAGEMENT OBJECT
44
INE
83
32a
32b
30b
32c
STATE PORTS OF DEVELOPERS
MGR
30a
01 D1
02 D2
03 D3
30c
85a
85b
82a
82b
82c
DEVELOPERS NEGOTIATION PORTS TUNED TO PHASE 1 OF THE CLOCK
MANAGER'S NEGOTIATION PORT TUNED TO BASE PHASE 0 OF THE CLOCK
CLOCK FOR THE NEGOTIATION PROCESS WITH CLOCK ID = 4 FOR THE BASE PHASE AND CLOCK ID = 5 FOR PHASE 1. EACH NEGOTIATION PORT ID MUST BE UNIQUE.
86

FIG. 8.

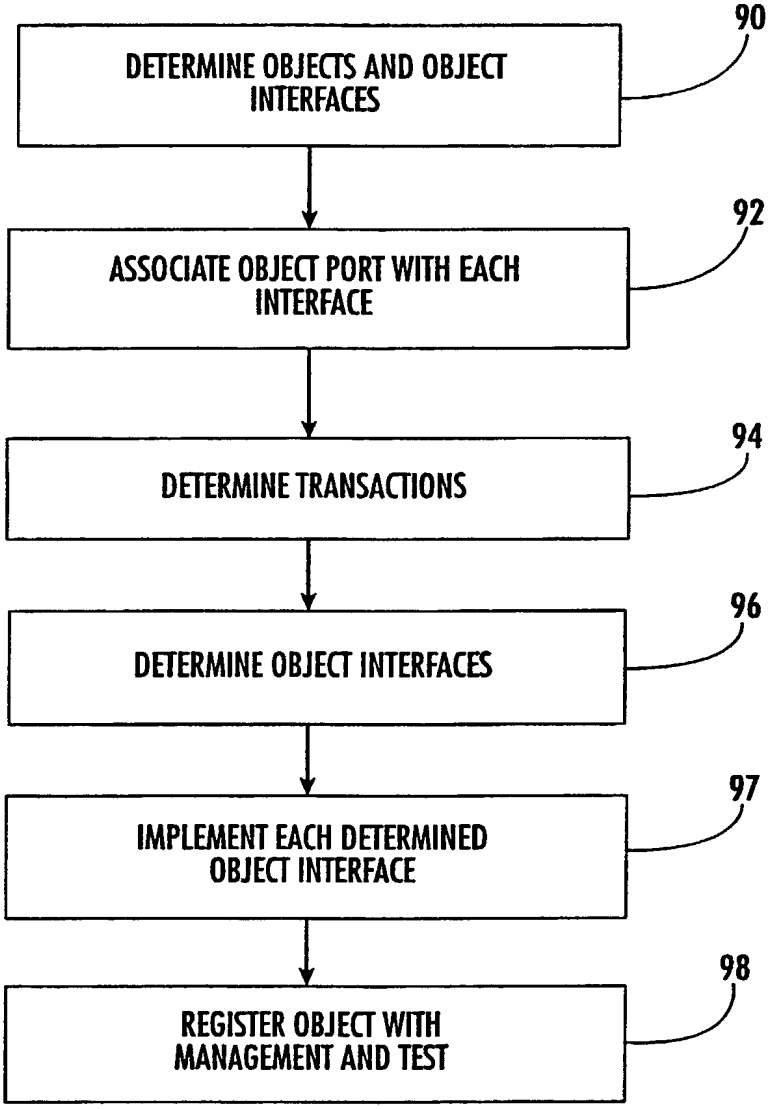

DETERMINE OBJECTS AND OBJECT INTERFACES
90
ASSOCIATE OBJECT PORT WITH EACH INTERFACE
92
DETERMINE TRANSACTIONS
94
DETERMINE OBJECT INTERFACES
96
IMPLEMENT EACH DETERMINED OBJECT INTERFACE
97
REGISTER OBJECT WITH MANAGEMENT AND TEST
98

FIG. 9.

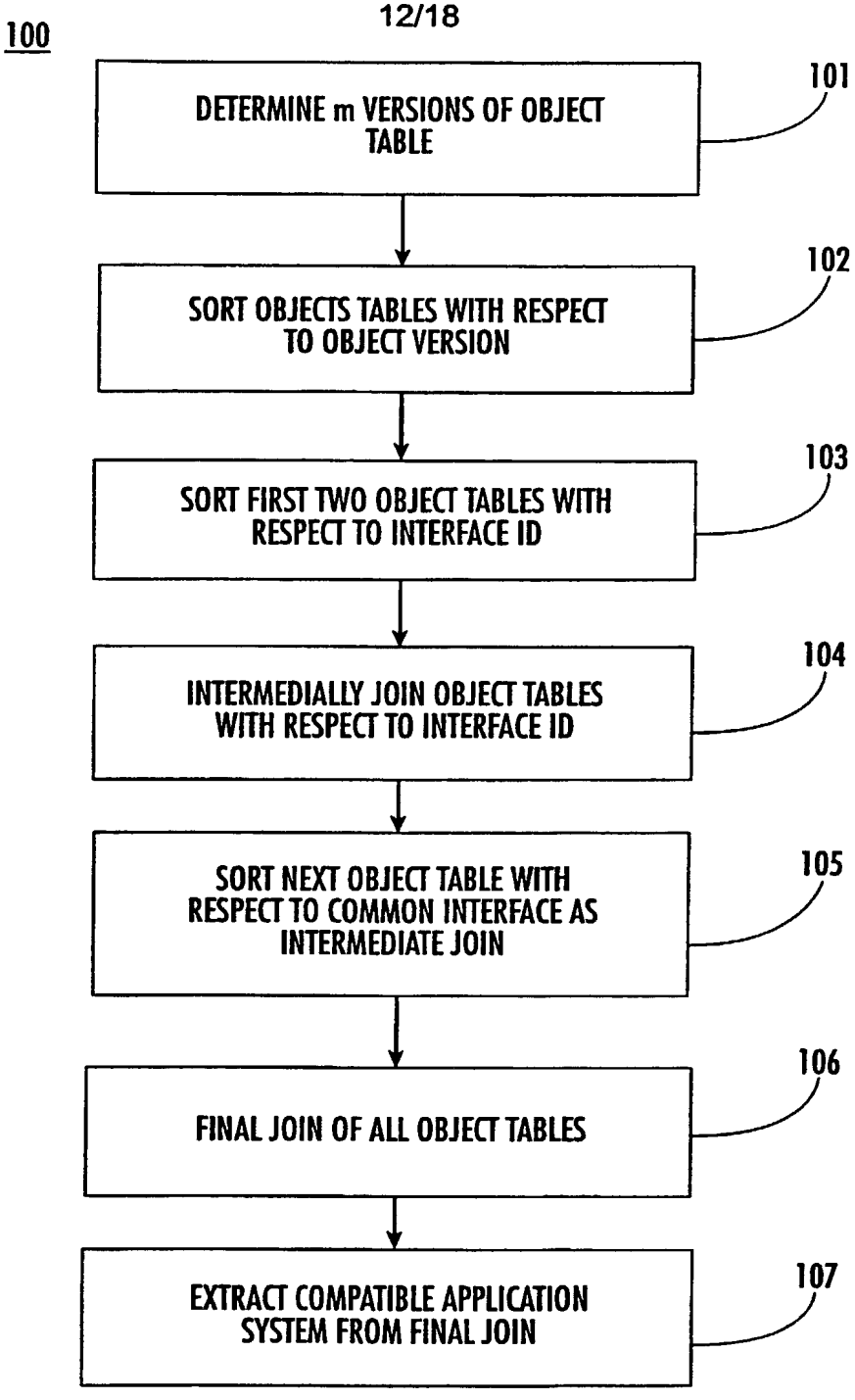

100
DETERMINE m VERSIONS OF OBJECT TABLE
101
SORT OBJECTS TABLES WITH RESPECT TO OBJECT VERSION
102
SORT FIRST TWO OBJECT TABLES WITH RESPECT TO INTERFACE ID
103
INTERMEDIALLY JOIN OBJECT TABLES WITH RESPECT TO INTERFACE ID
104
SORT NEXT OBJECT TABLE WITH RESPECT TO COMMON INTERFACE AS INTERMEDIATE JOIN
105
FINAL JOIN OF ALL OBJECT TABLES
106
EXTRACT COMPATIBLE APPLICATION SYSTEM FROM FINAL JOIN
107

FIG. 10.

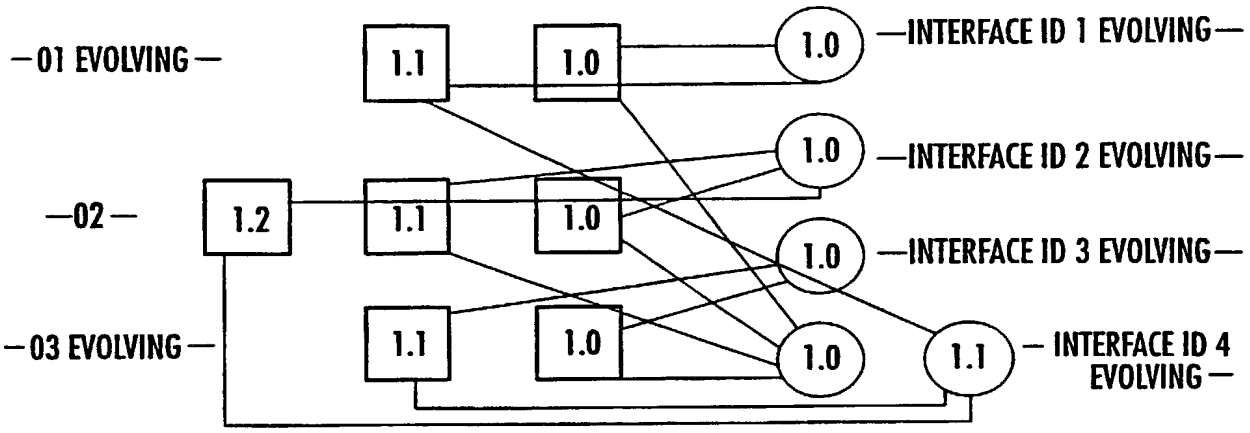


FIG. 11A.

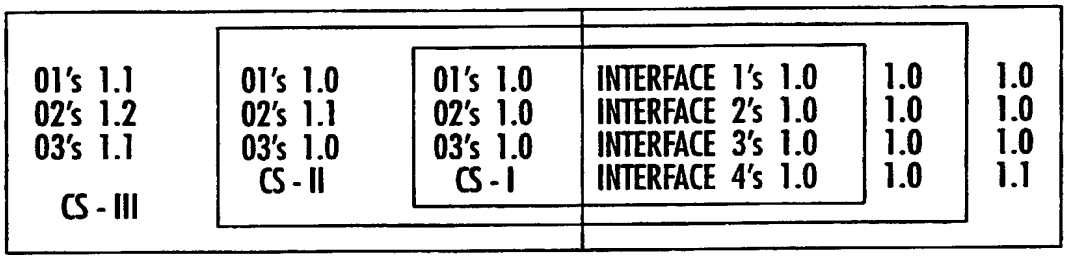


FIG. 11B.

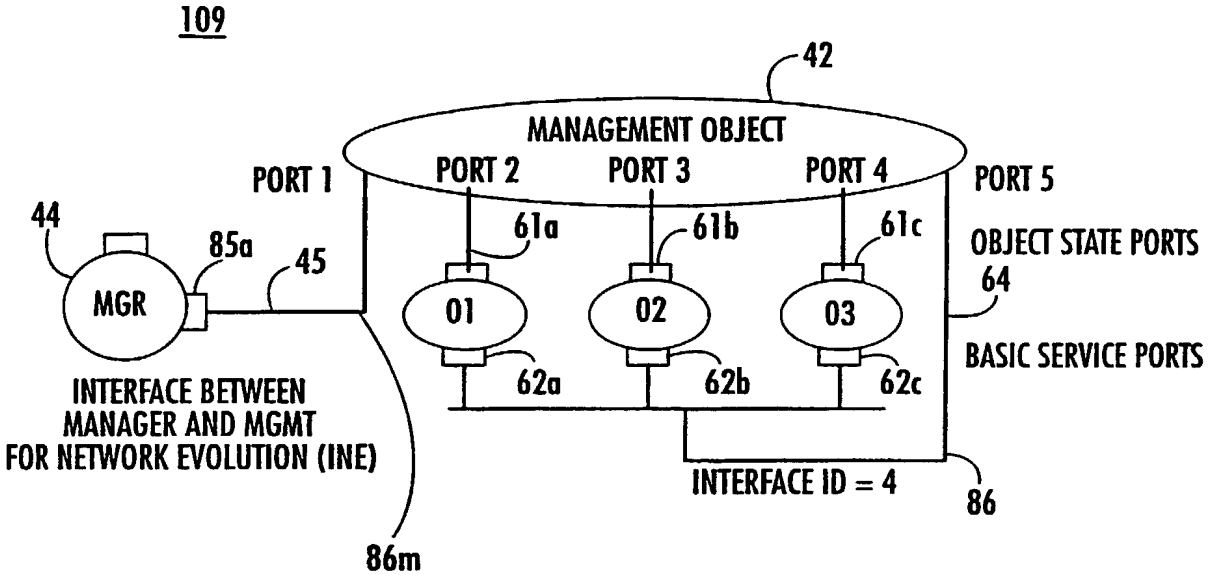

109
42
MANAGEMENT OBJECT
PORT 1
PORT 2
PORT 3
PORT 4
PORT 5
44
85a
45
61a
61b
61c
OBJECT STATE PORTS
64
MGR
01
02
03
BASIC SERVICE PORTS
62a
62b
62c
INTERFACE BETWEEN
MANAGER AND MGMT
FOR NETWORK EVOLUTION (INE)
INTERFACE ID = 4
86
86m

FIG. 12A.

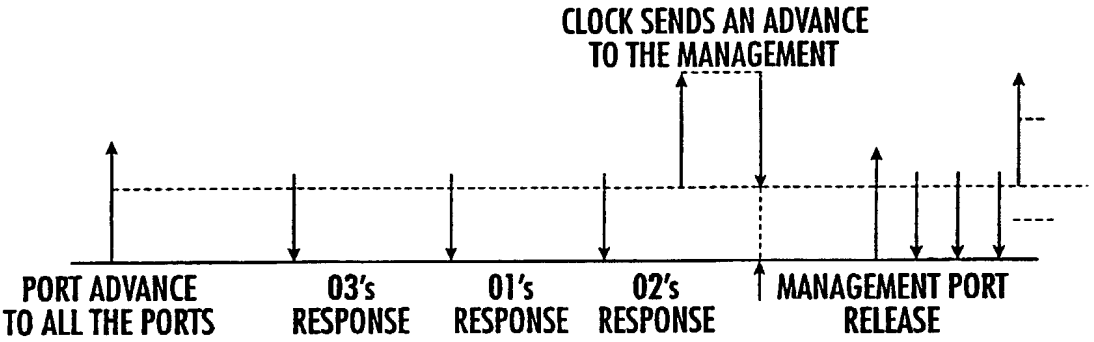

CLOCK SENDS AN ADVANCE
TO THE MANAGEMENT
PORT ADVANCE
TO ALL THE PORTS
03's
RESPONSE
01's
RESPONSE
02's
RESPONSE
MANAGEMENT PORT
RELEASE

FIG. 12B.

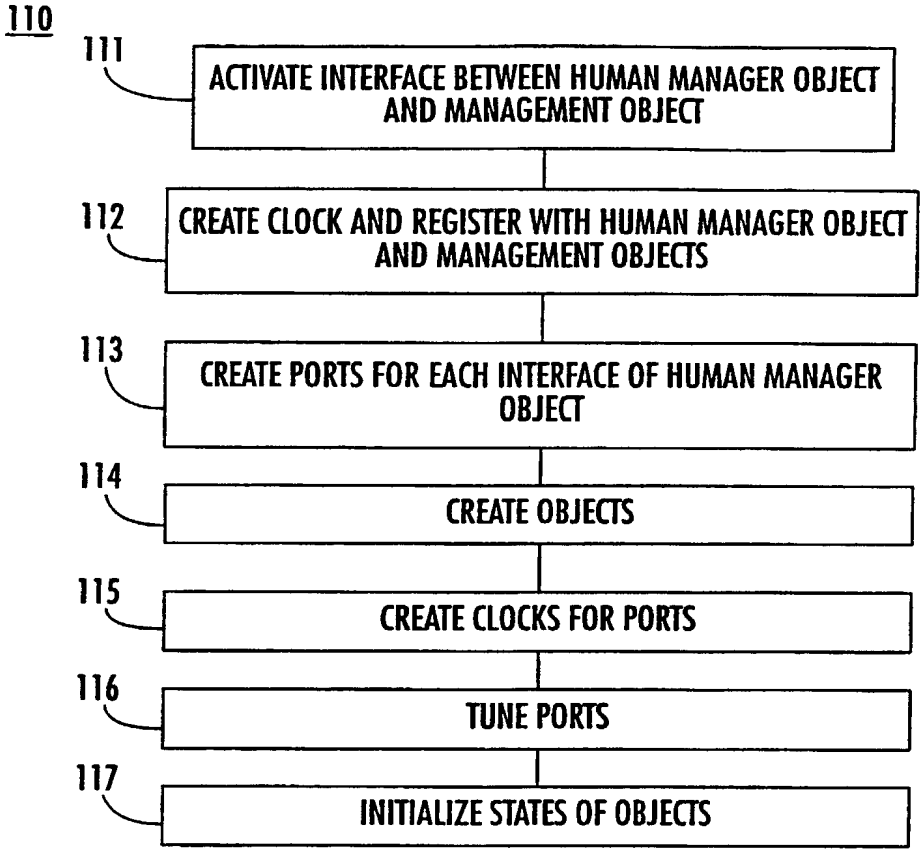

110
111
ACTIVATE INTERFACE BETWEEN HUMAN MANAGER OBJECT AND MANAGEMENT OBJECT
112
CREATE CLOCK AND REGISTER WITH HUMAN MANAGER OBJECT AND MANAGEMENT OBJECTS
113
CREATE PORTS FOR EACH INTERFACE OF HUMAN MANAGER OBJECT
114
CREATE OBJECTS
115
CREATE CLOCKS FOR PORTS
116
TUNE PORTS
117
INITIALIZE STATES OF OBJECTS

FIG. 13.

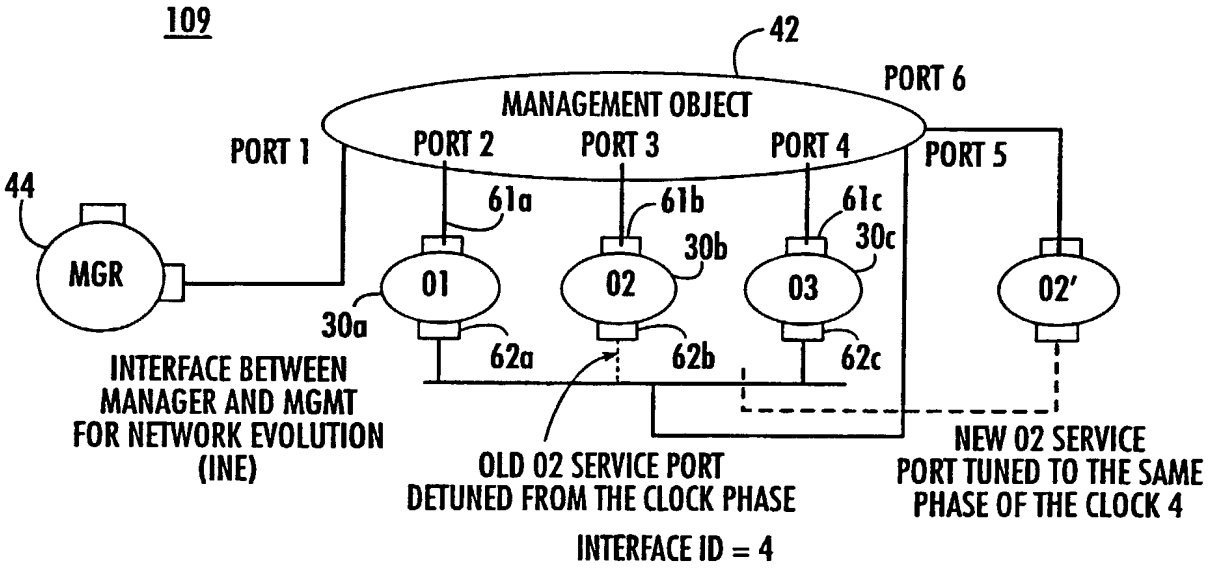

109
42
MANAGEMENT OBJECT
PORT 6
PORT 1
PORT 2
PORT 3
PORT 4
PORT 5
44
MGR
61a
61b
61c
30b
30c
01
02
03
02'
30a
62a
62b
62c
INTERFACE BETWEEN MANAGER AND MGMT FOR NETWORK EVOLUTION (INE)
OLD 02 SERVICE PORT DETUNED FROM THE CLOCK PHASE
NEW 02 SERVICE PORT TUNED TO THE SAME PHASE OF THE CLOCK 4
INTERFACE ID = 4

FIG. 14.

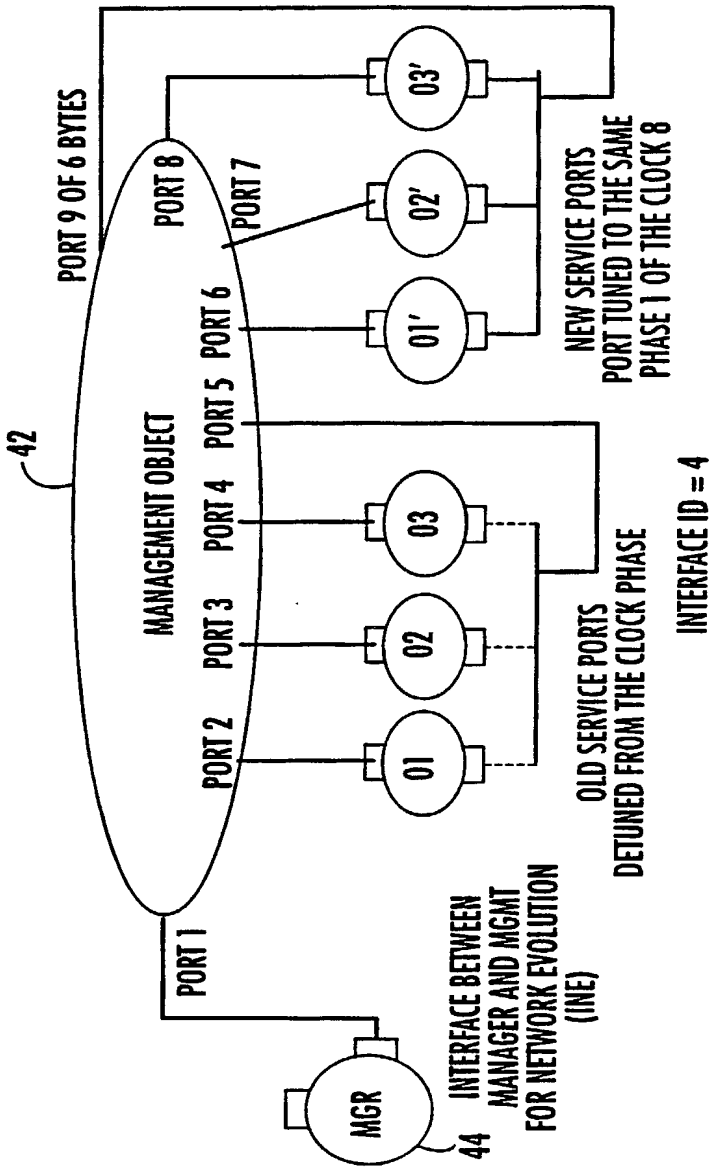

42
MANAGEMENT OBJECT
PORT 9 OF 6 BYTES
PORT 8
PORT 7
PORT 6
PORT 5
PORT 4
PORT 3
PORT 2
PORT 1
03'
02'
01'
03
02
01
NEW SERVICE PORTS
PORT TUNED TO THE SAME
PHASE 1 OF THE CLOCK 8
OLD SERVICE PORTS
DETUNED FROM THE CLOCK PHASE
INTERFACE ID = 4
MGR
44
INTERFACE BETWEEN
MANAGER AND MGMT
FOR NETWORK EVOLUTION
(INE)

FIG. 15.

Express Mail Label No.

Docket No.
1419-136 US

Declaration and Power of Attorney For Patent Application

English Language Declaration

As a below named inventor, I hereby declare that:

My residence, post office address and citizenship are as stated below next to my name,

I believe I am the original, first and sole inventor (if only one name is listed below) or an original, first and joint inventor (if plural names are listed below) of the subject matter which is claimed and for which a patent is sought on the invention entitled
Distributed Software Development Environment

the specification of which

(check one)

☐ is attached hereto.

☒ was filed on 28 April 2000 as United States Application No. or PCT International Application Number PCT/US00/11428

and was amended on _____ (if applicable)

I hereby state that I have reviewed and understand the contents of the above identified specification, including the claims, as amended by any amendment referred to above.

I acknowledge the duty to disclose to the United States Patent and Trademark Office all information known to me to be material to patentability as defined in Title 37, Code of Federal Regulations, Section 1.56.

I hereby claim foreign priority benefits under Title 35, United States Code, Section 119(a)-(d) or Section 365(b) of any foreign application(s) for patent or inventor's certificate, or Section 365(a) of any PCT International application which designated at least one country other than the United States, listed below and have also identified below, by checking the box, any foreign application for patent or inventor's certificate or PCT International application having a filing date before that of the application on which priority is claimed.

Prior Foreign Application(s)			Priority Not Claimed
(Number)	(Country)	(Day/Month/Year Filed)	☐
(Number)	(Country)	(Day/Month/Year Filed)	☐
(Number)	(Country)	(Day/Month/Year Filed)	☐

I hereby claim the benefit under 35 U.S.C. Section 119(e) of any United States provisional application(s) listed below:

60/131,506	29 April 1999
(Application Serial No.)	(Filing Date)
(Application Serial No.)	(Filing Date)
(Application Serial No.)	(Filing Date)

I hereby claim the benefit under 35 U. S. C. Section 120 of any United States application(s), or Section 365(c) of any PCT International application designating the United States, listed below and, insofar as the subject matter of each of the claims of this application is not disclosed in the prior United States or PCT International application in the manner provided by the first paragraph of 35 U.S.C. Section 112, I acknowledge the duty to disclose to the United States Patent and Trademark Office all information known to me to be material to patentability as defined in Title 37, C. F. R., Section 1.56 which became available between the filing date of the prior application and the national or PCT International filing date of this application:

(Application Serial No.)	(Filing Date)	(Status) (patented, pending, abandoned)
(Application Serial No.)	(Filing Date)	(Status) (patented, pending, abandoned)
(Application Serial No.)	(Filing Date)	(Status) (patented, pending, abandoned)

I hereby declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code and that such willful false statements may jeopardize the validity of the application or any patent issued thereon.

POWER OF ATTORNEY: As a named inventor, I hereby appoint the following attorney(s) and/or agent(s) to prosecute this application and transact all business in the Patent and Trademark Office connected therewith. *(list name and registration number)*

Bruce M. Collins, Reg. No. 20,066
Ronald Gould, Reg. No. 28,299
Diane Dunn McKay, Reg. No. 34,586
Timothy X. Gibson, Reg. No. 40,618
David P. Krivoshik, Reg. No. 39,258
Brian L. Buckwalter, Reg. No. 46,585

For the firm:
Mathews, Collins, Shepherd & Gould, P.A.
100 Thanet Circle, Suite 306
Princeton, NJ 08540
(609) 924-8555 Telephone
(609) 924-3036 Facsimile

Send Correspondence to: **Diane Dunn McKay, Esq.**
Mathews, Collins, Shepherd & Gould, P.A.
100 Thanet Circle, Suite 306
Princeton, NJ 08540

Direct Telephone Calls to: *(name and telephone number)*
Diane Dunn McKay 609-924-8555

Full name of sole or first inventor
Manjunath M. SURYANARAYANA

Sole or first inventor's signature — Date: May 17, 2002

Residence
Highland Park, New Jersey

Citizenship

Post Office Address
110A Cedar Lane

Highland Park, NJ 08904

Full name of second inventor, if any

Second inventor's signature — Date

Residence

Citizenship

Post Office Address